

设备管理

# 开发指南

文档版本 15  
发布日期 2022-06-14



版权所有 © 华为技术有限公司 2023。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暗示的声明或保证。

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

# 目录

<b>1 开发前必读（联通用户专用）</b> .....	<b>1</b>
<b>2 资源获取（联通用户专用）</b> .....	<b>5</b>
<b>3 从这里开始</b> .....	<b>10</b>
3.1 我是模组厂商（联通用户专用） .....	11
3.2 我是设备厂商（联通用户专用） .....	11
3.3 我是设备用户（联通用户专用） .....	14
3.4 我是应用开发者（联通用户专用） .....	16
<b>4 平台侧开发</b> .....	<b>18</b>
4.1 体验开发中心（联通用户专用） .....	18
4.2 创建项目（联通用户专用） .....	39
4.3 创建产品（联通用户专用） .....	42
4.4 开发 Profile.....	47
4.4.1 什么是 Profile（联通用户专用） .....	47
4.4.2 在线开发 Profile（联通用户专用） .....	48
4.4.3 离线开发 Profile（联通用户专用） .....	52
4.4.4 导出和导入 Profile.....	64
4.5 开发编解码插件.....	67
4.5.1 什么是编解码插件（联通用户专用） .....	67
4.5.2 在线开发插件（联通用户专用） .....	69
4.5.3 离线开发插件（联通用户专用） .....	133
4.5.4 下载和上传插件（联通用户专用） .....	163
4.6 调测产品（联通用户专用） .....	166
4.6.1 设备侧开发.....	172
4.6.1.1 使用 MQTTS 协议接入（联通用户专用） .....	172
4.6.1.2 使用 LoRaWAN 协议接入（联通用户专用） .....	173
4.6.1.3 使用 Modbus 协议接入（联通用户专用） .....	177
4.6.1.4 使用 Agent SDK 接入.....	183
4.6.1.4.1 Agent Lite SDK 使用指南（C）（联通用户专用） .....	183
4.6.1.4.2 Agent Lite SDK 使用指南（Java）（联通用户专用） .....	198
4.6.1.4.3 Agent Lite SDK 使用指南（Android）（联通用户专用） .....	209
4.6.1.5 使用模组接入.....	226
4.6.1.5.1 Agent Tiny SDK 使用指南（联通用户专用） .....	229

4.6.1.6 软固件升级调测.....	229
4.6.1.6.1 固件升级（联通用户专用）.....	229
4.6.1.6.2 软件升级（联通用户专用）.....	234
4.6.1.7 应用侧开发.....	240
4.6.1.7.1 使用 API 对接.....	240
4.6.1.7.1.1 API 使用指导（联通用户专用）.....	240
4.6.1.7.1.2 使用 Postman 调测（联通用户专用）.....	241
4.6.1.7.1.3 使用 Java API Demo 调测（联通用户专用）.....	259
4.6.1.7.2 使用 SDK 对接.....	272
4.6.1.7.2.1 Java SDK 使用指南（联通用户专用）.....	272
4.6.1.7.2.2 Python SDK 使用指南（联通用户专用）.....	285
<b>5 调测证书制作（联通用户专用）.....</b>	<b>299</b>
<b>6 自助测试（联通用户专用）.....</b>	<b>303</b>
<b>7 产品发布（联通用户专用）.....</b>	<b>312</b>
<b>8 商用对接（联通用户专用）.....</b>	<b>315</b>
<b>9 IoT 技术认证（联通用户专用）.....</b>	<b>323</b>

# 1 开发前必读（联通用户专用）

非联通用户请查看[设备接入服务](#)。

## 方案概述

基于设备管理服务去实现一个物联网解决方案时，需要完成以下开发操作：

开发操作	开发说明
<a href="#">平台侧的开发</a>	主要包括Profile的开发和编解码插件的开发。编解码插件的开发仅针对上报数据为二进制码流格式的设备，对于上报数据为JSON格式的设备不需要开发编解码插件。
<a href="#">设备侧的开发</a>	主要为设备与物联网平台的集成对接开发，包括设备接入物联网平台、业务数据上报和对平台下发控制命令的处理。
<a href="#">应用侧的开发</a>	主要为业务应用与物联网平台的集成对接开发，包括API接口的调用、业务数据的获取和HTTPS证书的管理。

基于物联网平台开发一个物联网解决方案的工作流程请参考[从这里开始](#)。

## 开发中心与设备管理服务的差异

- **了解开发中心**

开发中心是基于设备管理服务提供的物联网一站式开发工具，帮助开发者快速进行Profile（产品模型）和编解码插件的开发，同时提供在线自助测试、产品发布等多种能力，端到端指引物联网开发，帮助开发者提升集成开发效率、缩短物联网解决方案建设周期。

- 产品开发：提供产品开发向导，端到端引导开发者完成Profile开发、插件开发以及产品调测，助力物联网产品快速上线。
- 应用开发：支持对接信息、订阅调试、应用调试等能力，帮助开发者进行应用侧开发和调试，助力物联网应用便捷开发。
- 自助测试：支持对设备、应用进行自动化测试，并生成测试报告，检验产品是否达到发布标准。
- 产品发布：产品在开发中心完成自助测试后，开发者可以一键申请发布到产品中心，已发布的产品可直接应用于商用环境。

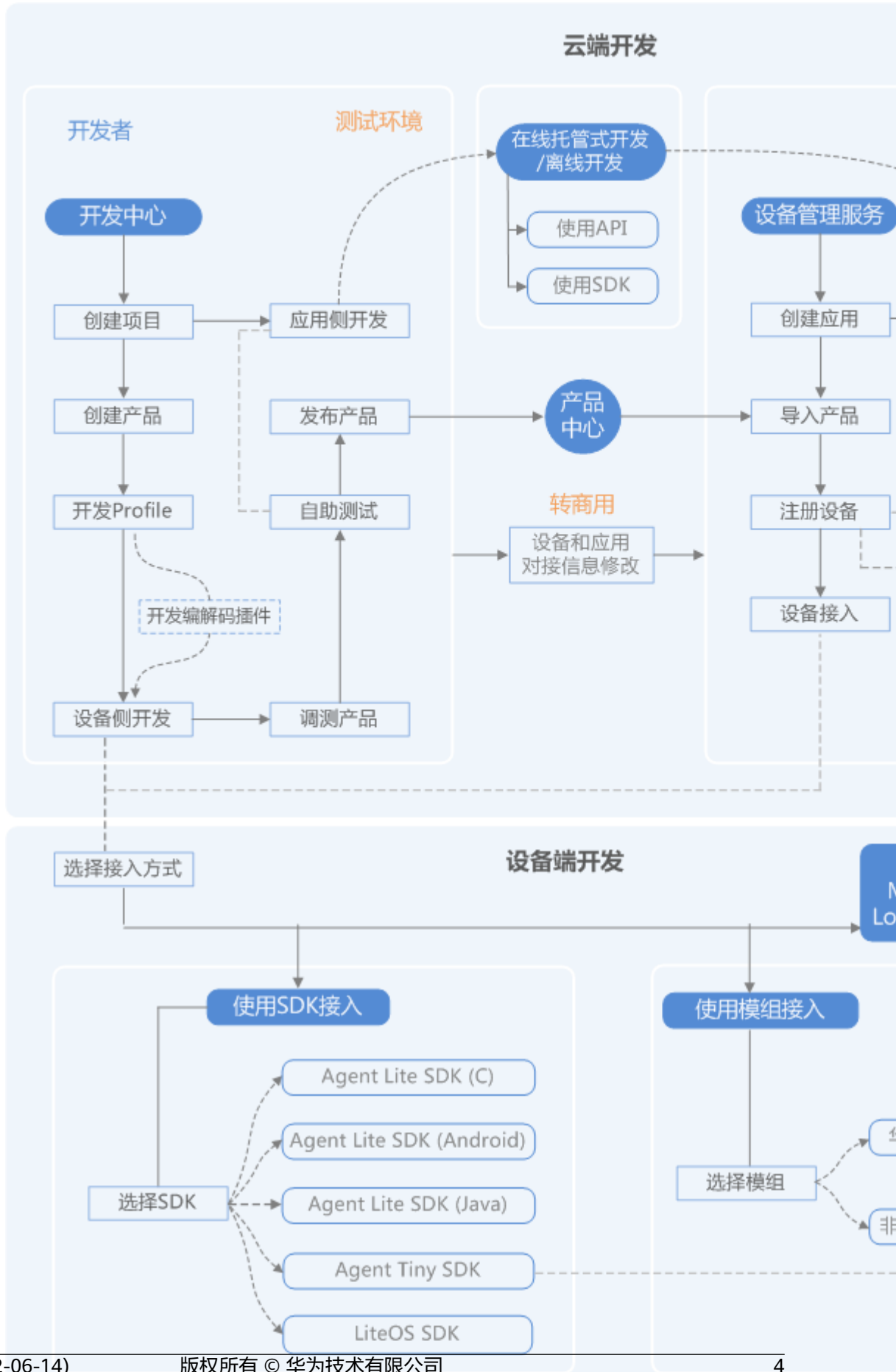
- **了解设备管理服务**  
请前往[平台简介](#)查看设备管理服务的介绍。
- **开发中心和设备管理服务有哪些差异？**
  - a. 两者所在的物联网平台环境不一样  
开发中心所在的平台环境为测试环境，设备管理服务所在的平台环境为商用环境。两个环境的设备数据不能互通，已对接测试环境的设备/应用要迁移商用环境，需要在商用环境重新创建应用和注册设备，并修改设备/应用中平台的接入信息。
  - b. 两者的设备数量限制不一样  
开发中心单个项目（应用）下最多可注册20个设备，设备管理服务单个应用下最多可注册1000万个设备。在其他的平台能力上，两者保持一致，例如开发中心和设备管理服务使用同样的API，更多的平台能力使用限制请查看[使用限制](#)。
  - c. 两者的计费策略不一样  
开发中心为免费使用，设备管理为付费使用，设备管理服务的具体计费策略请查看[计费详情](#)。
- **开发中心和设备接入服务是否有联系？**  
开发中心是基于设备管理服务的物联网开发工具，在开发中心上开发的产品（Profile文件和编解码插件）只适用于设备管理服务，不能在设备接入服务中导入和使用。

## 业务概览

当用户在开通设备管理服务时，系统默认一起开通设备接入服务，即用户在使用设备管理服务时，包含设备接入服务的能力。使用设备管理服务的完整流程如下图所示，主要分为在开发中心（测试环境）进行产品开发、在控制台（商用环境）进行上线和日常管理。

- **产品开发：**开发者在进行设备接入前，基于开发中心进行相应的开发工作，包括平台侧开发、设备侧开发、应用侧开发，是真实设备接入到设备管理服务的前提条件。
- **上线（转商用）：**基于设备管理服务提供的控制台，将真实设备接入到设备管理服务中，并对接用户开发的应用服务器，实现设备的远程监控和控制。
- **日常管理：**真实设备接入后，基于控制台或者API接口，进行日常的应用管理和设备管理。

远端开发	转商用	设备侧开发	云端日常管理
<ul style="list-style-type: none"> <li>● 创建项目</li> <li>● 创建产品</li> <li>● 开发Profile</li> <li>● 开发编解码插件</li> <li>● 设备侧开发</li> <li>● 应用侧开发                             <ul style="list-style-type: none"> <li>- API使用指导</li> <li>- 使用SDK对接</li> </ul> </li> <li>● 调测产品</li> <li>● 自助测试</li> <li>● 产品发布</li> </ul>	<ul style="list-style-type: none"> <li>● 创建应用</li> <li>● 导入产品</li> <li>● 注册设备</li> <li>● 接入设备</li> <li>● 应用接入</li> <li>● 对接验证</li> </ul>	<ul style="list-style-type: none"> <li>● 使用MQTT协议接入（联通用户专用）</li> <li>● 使用Modbus协议接入（联通用户专用）</li> <li>● 使用Agent SDK接入</li> <li>● 使用模组接入</li> <li>● 软固件升级调测</li> </ul>	<ul style="list-style-type: none"> <li>● 应用管理                             <ul style="list-style-type: none"> <li>- 订阅推送</li> <li>- 授权访问</li> </ul> </li> <li>● 设备管理                             <ul style="list-style-type: none"> <li>- 设备注册鉴权</li> <li>- 数据上报</li> <li>- 命令下发</li> <li>- 设备配置更新</li> <li>- 设备影子</li> <li>- 规则引擎</li> <li>- 群组与标签</li> <li>- 设备监控</li> <li>- 远程诊断</li> <li>- 固件升级</li> <li>- 软件升级</li> <li>- 网关与子设备</li> </ul> </li> </ul>





# 2 资源获取（联通用户专用）

## 平台对接信息

设备和应用接入物联网平台前需要获取平台的接入地址信息。

平台环境	获取途径																								
开发中心（测试环境）	<p>进入<b>开发中心</b>的具体项目中，在“应用 &gt; 对接信息”页面查看“设备接入信息”和“应用接入信息”。</p> 																								
设备管理服务（商用环境）	<p>在物联网平台的<b>管理控制台</b>，选择设备接入，在“设备和应用接入信息”下可查看设备和应用的接入地址信息。</p>  <table border="1"> <thead> <tr> <th>类型</th> <th>域名</th> <th>端口</th> <th>协议</th> </tr> </thead> <tbody> <tr> <td>应用对接信息</td> <td>iot-<span style="background-color: #ccc;">                    </span>.myhuaweicloud.com</td> <td>8743</td> <td>HTTPS</td> </tr> <tr> <td>设备对接信息</td> <td>iot-<span style="background-color: #ccc;">                    </span>.myhuaweicloud.com</td> <td>5683</td> <td>CoAP</td> </tr> <tr> <td>设备对接信息</td> <td>iot-<span style="background-color: #ccc;">                    </span>.myhuaweicloud.com</td> <td>5684</td> <td>CoAPS</td> </tr> <tr> <td>设备对接信息</td> <td>iot-<span style="background-color: #ccc;">                    </span>.myhuaweicloud.com</td> <td>8943</td> <td>HTTPS</td> </tr> <tr> <td>设备对接信息</td> <td>iot-<span style="background-color: #ccc;">                    </span>.myhuaweicloud.com</td> <td>8883</td> <td>MQTTS</td> </tr> </tbody> </table>	类型	域名	端口	协议	应用对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8743	HTTPS	设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	5683	CoAP	设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	5684	CoAPS	设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8943	HTTPS	设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8883	MQTTS
类型	域名	端口	协议																						
应用对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8743	HTTPS																						
设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	5683	CoAP																						
设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	5684	CoAPS																						
设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8943	HTTPS																						
设备对接信息	iot- <span style="background-color: #ccc;">                    </span> .myhuaweicloud.com	8883	MQTTS																						

## 设备开发资源

物联网平台支持设备通过MQTT协议和LWM2M/CoAP协议进行接入，设备可以通过调用设备侧的接口或者集成SDK的方式接入到物联网平台。

资源包名	描述	下载路径
Agent Lite SDK(Linux C)	设备可以通过集成Agent Lite SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。 使用指导可以参考 <a href="#">Agent Lite API参考(C)</a> 和 <a href="#">Agent Lite SDK使用指南(C)</a> 。	<ul style="list-style-type: none"> <li>• <a href="#">Agent Lite SDK(Linux通用)</a></li> <li>• <a href="#">Agent Lite SDK(指定工具链)</a></li> <li>• <a href="#">Agent Lite Demo(C-Linux)</a></li> </ul>
Agent Lite SDK(Windows C)	设备可以通过集成Agent Lite SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。 使用指导可以参考 <a href="#">Agent Lite API参考(C)</a> 和 <a href="#">Agent Lite SDK使用指南(C)</a> 。	<ul style="list-style-type: none"> <li>• <a href="#">Agent Lite SDK(Windows)</a></li> <li>• <a href="#">Agent Lite Demo(C-Windows)</a></li> </ul> <p><b>说明</b> Demo的libs文件夹下为32位SDK，若需要使用64位SDK，请下载并替换Demo中的SDK。</p>
Agent Lite SDK(Java)	设备可以通过集成Agent Lite SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。 使用指导可以参考 <a href="#">Agent Lite API参考(Java)</a> 和 <a href="#">Agent Lite SDK使用指南(Java)</a> 。	<p><a href="#">Agent Lite Demo(Java)</a></p> <p><b>说明</b></p> <ul style="list-style-type: none"> <li>• SDK在Demo的libs文件夹下。</li> <li>• Agent Lite Demo(Java)仅适用于Windows系统环境。</li> </ul>
Agent Lite SDK(Android)	设备可以通过集成Agent Lite SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。 使用指导可以参考 <a href="#">Agent Lite API参考(Android)</a> 和 <a href="#">Agent Lite SDK使用指南(Android)</a> 。	<p><a href="#">Agent Lite Demo(Android)</a></p> <p><b>说明</b> SDK在Demo的libs文件夹下。</p>
LiteOS SDK	设备可以通过集成LiteOS SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。	<a href="#">LiteOS SDK</a>

资源包名	描述	下载路径
Profile模板	Profile模板中包含了典型场景的Profile样例，开发者可以在模板基础进行修改，定义自己需要的Profile。 使用指导可以参考 <a href="#">离线开发Profile</a> 。	<a href="#">Profile开发示例</a>
编解码插件样例	编解码插件的代码样例工程，开发者可以基于该样例工程进行二次开发。 使用指导可以参考 <a href="#">离线开发插件</a> 。	<a href="#">编解码插件开发样例</a>
编解码插件检测工具	用于检测离线开发的编解码插件的编解码能力是否正常。	<a href="#">编解码插件检测工具</a>
NB-IoT设备模拟器	用于模拟以CoAP/LWM2M协议接入物联网平台的NB设备，实现数据上报和命令下发功能。 使用指导可以参考 <a href="#">体验开发中心</a> 。	<a href="#">NB-IoT设备模拟器</a>
IoT Studio	IoT Studio是基于LiteOS嵌入式系统软件开发的工具，支持C、C++、汇编等多种开发语言，提供了代码编辑、编译、烧录及调试等一站式开发体验。 使用指导可以参考 <a href="#">设备侧开发实践</a> 。	<a href="#">IoT Studio</a>

## 应用开发资源

为了降低应用的开发难度、提升开发效率，物联网平台开放的丰富的Restful API和SDK包。应用通过调用物联网平台的API，实现安全接入、设备管理、数据采集、命令下发等业务场景。

资源包名	描述	下载
应用侧开发 API JAVA Demo	物联网平台为应用服务器提供了Restful API，能够让开发者快速验证Restful接口开放的能力，体验业务功能，熟悉业务流程。 使用指南可以参考 <a href="#">IoT平台应用侧API参考</a> 。	<a href="#">API JAVA Demo</a>
应用侧开发 Java SDK	Java SDK提供JAVA方法调用物联网平台Restful接口与平台通信，Demo提供调用SDK接口的样例代码。 使用指南可以参考 <a href="#">IoT平台应用侧JAVA SDK API参考</a> 和 <a href="#">JAVA SDK 使用指南</a>	<ul style="list-style-type: none"> <li>• <a href="#">JAVA SDK</a></li> <li>• <a href="#">JAVA SDK Demo</a></li> </ul>
应用侧开发 Python SDK	Python SDK提供Python方法调用平台Restful接口与平台通信，Demo提供调用SDK接口的样例代码。 使用指南可以参考 <a href="#">IoT平台应用侧Python SDK API参考</a> 和 <a href="#">Python SDK使用指南</a> 。	<ul style="list-style-type: none"> <li>• <a href="#">Python SDK</a></li> <li>• <a href="#">Python SDK Demo</a></li> </ul>

## 证书资源

在设备和应用对接物联网平台的部分场景中，需要在设备侧和应用侧集成相应证书。请点击获取[证书文件](#)。

### 说明

此证书文件只适用于华为云物联网平台。

证书类型，证书格式，以及适用开发语言，用途详见下表。

证书包名称	一级目录	二级目录	三级目录	说明
certificate	Northbound API	code	Java	应用服务器通过HTTPS协议调用物联网平台接口，用于校验物联网平台的合法性时，使用该目录下的证书。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	
		postman	-	Postman通过HTTPS协议调试物联网平台接口时，使用该目录下的证书。

证书包名称	一级目录	二级目录	三级目录	说明
	Agent Lite	Android	-	终端设备或网关通过集成Agent Lite SDK接入物联网平台时，使用该目录下的证书。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	-	

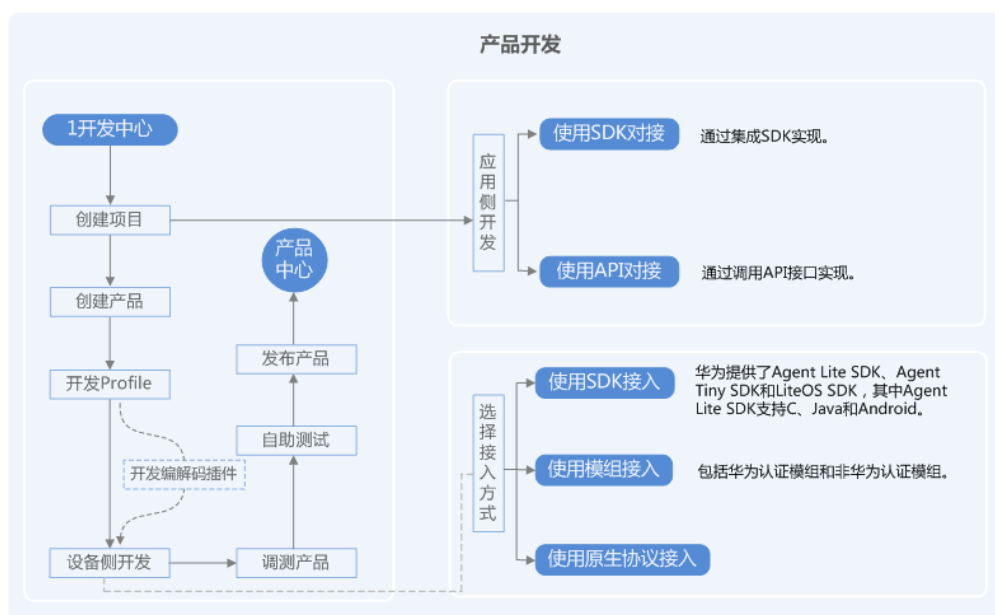
# 3 从这里开始

**联通用户专用，非联通用户请查看[设备接入服务](#)。**

使用设备管理服务的完整流程如下图所示，主要分为产品开发、上线和日常管理三个部分。用户可以根据自己角色的不同，执行对应的操作即可。

1. 产品开发：在开发中心（测试环境）操作，开发者在接入设备前，需要进行相应的开发工作，包括[平台侧开发](#)（开发中心）、[设备侧开发](#)、[应用侧开发](#)，是真实设备接入到设备管理服务的前提条件。

其中设备侧开发和应用侧开发不分先后顺序，可同步进行。



2. 上线（转商用）：基于设备管理服务提供的控制台，将真实设备接入到设备管理服务中，并对接用户开发的应用服务器，实现设备的远程监控和控制。
3. 日常管理：真实设备接入后，基于控制台或者API接口，进行日常的应用管理和设备管理，详情请参考[使用指南](#)。

- 3.1 [我是模组厂商（联通用户专用）](#)
- 3.2 [我是设备厂商（联通用户专用）](#)
- 3.3 [我是设备用户（联通用户专用）](#)
- 3.4 [我是应用开发者（联通用户专用）](#)

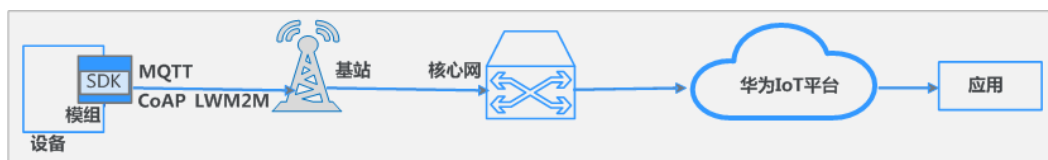
## 3.1 我是模组厂商（联通用户专用）

非联通用户请查看[设备接入服务](#)。

芯片是一种把集成电路小型化的方式，并时常制造在半导体晶圆表面上。模组是由若干个显示模块、驱动电路、控制电路、芯片以及相应的结构件构成的一个独立的显示单元。模组与芯片关系就像是U盘和Flash芯片的关系，用户99%的情况是直接使用模组，没有人直接用芯片。当前模组厂商主要提供Wifi、NB-IoT、2/3/4/5G等通信模组。

在物联网解决方案中，作为模组厂商的您需要让模组实现MQTTs、LWM2M、CoAP等物联网协议栈及连接平台的能力，您只需要将Agent Tiny SDK集成在现有的模组固件包中，这样模组就具备了接入华为物联网平台的能力。我们将提供[华为IoT技术认证](#)支持，帮助您快速完成SDK集成，通过华为认证的模组可以申请[入驻华为云市场](#)，我们将会推荐给物联网平台用户。查询[华为已认证的模组](#)。

Agent Tiny SDK具有普适性，可以广泛移植于WiFi模组、2/3/4/5G模组、NB-IoT模组，无需考虑模组类型（[了解Agent Tiny SDK详情](#)）。如何在模组中集成Agent Tiny SDK，请参考[华为IoT技术认证流程](#)，获取相关技术支持。



## 3.2 我是设备厂商（联通用户专用）

非联通用户请查看[设备接入服务](#)。

### 角色介绍

作为开发并销售最终设备的厂商，您需要进行设备集成开发，以便让设备具备接入物联网平台的能力。我们将提供[华为IoT技术认证](#)支持，帮助您快速完成设备适配和认证。

通过华为认证的设备可以申请[入驻华为云市场](#)，我们将会推荐给物联网平台用户。查询：[华为已认证的设备](#)。

根据设备是否具有IP通信能力，设备接入平台分为以下两种方式：[设备直接接入平台](#)和[设备通过网关接入平台](#)，您可以根据自己的设备选择合适的方式。

### 前提条件

已完成[平台侧开发](#)。

### 直接接入平台

针对已实现TCP/IP协议栈的设备，可以直接与平台进行通信，常见的设备包括网关、以太网设备、NB-IoT设备等。根据设备自身硬件的特点不同，华为物联网平台提供四种方式直接接入，您需要根据自身行业特征及业务情况选择合适的接入方案，四种方案优缺点如下：

接入方式	方案优点	方案约束
方案一：集成 LiteOS接入物联网平台	有对应的SDK，集成难度较低，对您的开发技能要求较低。	方案不灵活，开发者只能根据LiteOS提供的API去实现自己的功能，超出API外的功能，无法自定义，对于设备的硬件要求比较高。
方案二：没有配置模组时，通过集成Agent Lite SDK接入物联网平台	有对应的SDK，集成难度低，对您的开发技能要求较低。	方案不灵活，开发者只能根据Agent SDK提供的API去实现自己的功能，超出API外的，无法自定义。
方案三：配置模组时，通过集成Agent Tiny SDK接入物联网平台	集成难度非常低，对您的开发技能要求低。	需要采购指定型号的模组。
方案四：通过实现原生协议接入物联网平台	方案比较灵活，可根据业务需要，实现协议定义的功能，对设备硬件无限制。	需要从底层协议开始实现，集成难度大，代码开发量大，对于开发者要求高。

- **方案一：**设备需要具备智能操作系统，硬件满足RAM容量 > 32KB，Flash容量 > 128KB时，通过集成LiteOS操作系统接入平台。

集成开发流程详见以下链接，请根据设备支持的协议选择：

- [LiteOS SDK端云互通组件CoAP/LWM2M开发指南](#)



- **方案二：**设备没有配置模组时，需通过集成Agent Lite SDK接入平台。[了解Agent Lite SDK详情](#)。

当设备存储及计算能力较强（满足RAM容量 > 4MB，Flash容量 > 2MB）时，推荐设备集成Agent Lite SDK。

目前Agent Lite SDK支持C、Java和Android三个版本。实际开发中，请根据开发时使用的语言、平台，选用合适的设备端SDK集成：

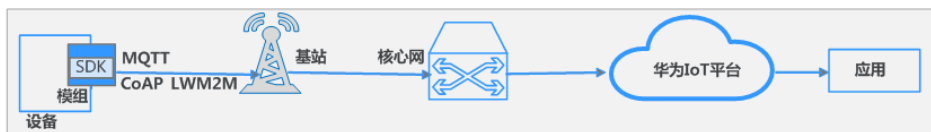
- [Agent Lite SDK集成开发指导（C）](#)
- [Agent Lite SDK集成开发指导（Java）](#)
- [Agent Lite SDK集成开发指导（Android）](#)



- **方案三：**当设备配置有模组时，根据模组特征，选择性集成Agent Tiny SDK接入平台。[了解Agent Tiny SDK详情](#)



- 如果采用经过华为认证的模组，该模组已集成Agent Tiny SDK，可以利用模组直接接入物联网平台。您可以访问[华为云市场](#)，购买已认证的模组。



- 如果采用未经过华为认证的模组，您需要集成Agent Tiny SDK，根据SDK集成位置，分为两种模式：

- **MCU+模组模式：**此模式下，设备包含MCU（Microcontroller Unit）和通信模组，其中MCU集成 Agent Tiny SDK及运行产品逻辑，模组作为通信模块，提供通信网络。请参考[华为IoT技术认证流程](#)，获取相关技术支持。



- **OpenCPU模式：**此模式下，设备只包含通信模组，模组集成 Agent Tiny SDK及运行产品逻辑，请参考[华为IoT技术认证流程](#)，获取相关技术支持。



- **方案四：**设备通过实现原生MQTT协议接入，适合一切设备。可查看[《设备集成》](#)了解操作。您可访问[《MQTT接口参考》](#)获取华为物联网平台开放的能力。



## 通过网关接入平台

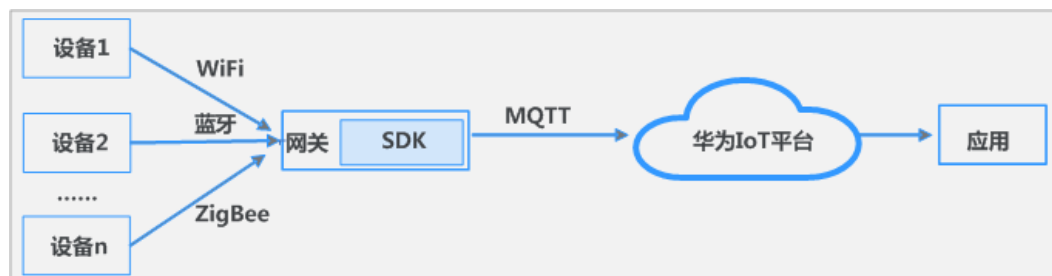
针对未实现TCP/IP协议栈的设备，由于无法直接同物联网平台通信，它需要通过网关进行数据转发。常见设备包括智慧园区中的照明系统、楼宇系统设备等。这些场景下的限制如下：

制约因素	详细描述
设备限制	设备非常简单，没有集成TCP/IP协议栈，无法提供IP通信网络能力，只能基于一些简单的近场通信协议如ZigBee、ZWave、Bluetooth或者是其他的一些非IP有线方式传输协议如串口、并口等接入，此时需要一个网关设备，先让设备接入到网关，再通过网关与华为物联网平台通信。
网络限制	设备部署在独立的网络内，由于安全等考虑，无法直接与物联网平台互通，需要借助网关进行网络桥接。
业务限制	下层设备的管理控制逻辑比较复杂，需要利用网关软件自身具有对应的逻辑控制能力配合物联网平台进行业务管理。

制约因素	详细描述
集成限制	子设备已经接入到现有系统，不愿意再进行设备改造，需要将现有系统通过网关接入到物联网平台。

针对此类设备，通过在网关上集成Agent Lite SDK, 设备将数据上报给网关，通过网关转发到华为物联网平台。华为公司协同合作伙伴，对已集成Agent Lite SDK的网关进行了认证，您可以访问[华为云市场](#)，购买符合自身业务的网关产品。目前Agent Lite SDK支持C、Java和Android三个版本。实际开发中，请根据开发时使用的语言、平台，选用合适的SDK集成：

- [Agent Lite SDK集成开发指导（C）](#)
- [Agent Lite SDK集成开发指导（Java）](#)
- [Agent Lite SDK集成开发指导（Android）](#)



### 3.3 我是设备用户（联通用户专用）

非联通用户请查看[设备接入服务](#)。

作为设备用户，您既是设备的购买者，又可能是设备的实际管理者，通常您需要考虑两件事：

1. 在采购设备时，如何评估设备满足您的业务需求并且具备接入物联网平台的能力。关于此方面的知识，您可以参考[设备能力评估](#)。
2. 在进行日常管理时，针对不具备接入物联网平台的设备，如何将现有设备进行改造，以便设备能接入到物联网平台中。关于此方面的知识，您可以参考[设备改造](#)。

#### 前提条件

已完成[平台侧开发](#)。

#### 设备能力评估

本节主要关注设备通信能力的评估。由于承接业务的不同，对于设备的要求也不同，采购的设备通信能力评估一般由以下几个流程：



1. 针对您所处的行业，您需要深度分析业务对于设备（硬件资源、电池、性能等）、网络实时性及网络覆盖度等要求，确保现有设备满足业务需求。典型案例如下表所示。

典型行业	设备特征	技术关注点
智慧抄表	接入设备数量多、电量有限、设备常常分布在地上或地下多个区域，要求有信号覆盖。	设备数量多，接入成本要低，自身硬件资源有限，要求设备网络要深度覆盖，低功耗，数据量较少。
智慧园区	应用子系统多、设备传感器种类多、无统一的通信协议标准，上报的数据无统一格式。设备大部分无IP通信能力，需要通过网关接入。可通过有线网络接入。	应用独立子系统多，格式不统一，数据孤岛多，需要一个统一平台管理。设备不具备IP通信能力，硬件资源有限，通常通过网关转发数据，因此主要关注网关的通信能力。
车联网	设备电量充足。	需要实时对车辆数据进行分析等，因此需要接入网络实时性高，数据传输速率高等，主要关注接入网络选型，要选传输速率较高的无线网络接入。

2. 根据业务对于数据及成本要求，确认设备支持的网络是否能够满足业务需求。

业务应用	推荐的接入网络	网络特点
智慧家庭、智慧楼宇等	以太网	传输速率可达到10Gbit/s、随时在线、成本高。
车联网、视频监控等	4G/5G/LTE-v	传输速率>10Mbit/s、功耗高、成本高。
电子广告、无线ATM、梯联网等	eMTC、GPRS	传输速率<1Mbit/s、功耗较低、成本较低。
远程抄表、智能停车等	NB-IoT、LoRa	传输速率<100Kbit/s、功耗低、成本低、穿透力强、信号覆盖好。

3. 确认设备是否集成了物联网协议，以便保证设备可以连接到物联网平台，利用物联网平台进行设备管理。华为物联网平台目前支持设备采用以下协议接入。

通信协议	协议描述	应用场景
LWM2M	LWM2M是开发移动联盟OMA定义的用于设备管理的应用层通讯协议，主要使用在资源受限的嵌入式设备上。	NB-IoT设备接入平台，业务实时性要求不高，低功耗、信号广覆盖场景。
CoAP	CoAP是资源受限设备和受限网络专用的Web传输协议，专为机器对机器的应用而设计。CoAP提供请求/响应交互模型，支持内置的服务和资源发现。需要底层实现UDP协议。	NB-IoT设备接入平台，业务实时性要求不高，低功耗、信号广覆盖场景。
MQTT	MQTT是一种物联网连接协议，提供非常轻量级的发布/订阅消息传输方式，用于在低带宽、不可靠的网络的设备管理。该协议构建于TCP/IP协议上。	对设备的可靠性和实时性要求高，适合长连接的场景，如智能路灯等。

4. 根据前面步骤，总结设备的特征，选择与业务相匹配的设备。例如针对智能抄表行业，要求电表通信模块具有覆盖广、穿透力强、耗电量小、成本低特点，因此接入网络选择NB-IoT网络，设备采用NB-IoT模组，模组集成了LWM2M协议，可以保证设备接入华为物联网平台。

## 设备改造

设备如果不具备接入华为物联网平台能力，需要对设备进行改造，改造方法跟设备厂商进行设备集成方法一样，详见[我是设备厂商](#)介绍。

## 3.4 我是应用开发者（联通用户专用）

非联通用户请查看[设备接入服务](#)。

华为物联网平台面向全球各行各业提供物联网服务，并通过Restful API的形式对外开放物联网平台丰富的设备管理能力。应用开发人员基于API接口开发所需的行业应用，如智慧城市、智慧园区、智慧工业、车联网等行业应用，满足不同行业的需求。开发包括如下几个关键阶段。



应用服务器开发阶段	操作指引
调用API接口或者集成SDK	应用开发者通过 <a href="#">调用API接口</a> 或者 <a href="#">集成SDK</a> 调用物联网平台的能力进行应用服务器的开发。可参考 <a href="#">表3-1</a> 选择合适的开发方式。

应用服务器开发阶段	操作指引
开发接收设备数据的接口	应用服务器需开发接收数据对应的接口（对应订阅管理接口中的callbackUrl地址），用于应用服务器向物联网平台发起订阅后，接收物联网平台推送的设备相关数据。
制作调测证书与加载	<ul style="list-style-type: none"> <li>应用服务器通过HTTPS协议调用物联网平台提供的API接口，需要在应用服务器上预置CA证书，用于应用服务器校验物联网平台的合法性，该证书可通过下载<a href="#">证书文件</a>获取。</li> <li>物联网平台通过HTTPS或HTTP协议向应用服务器推送数据，当使用HTTPS协议时需要在物联网平台上加载CA证书，同时在应用服务器上加载设备证书，用于物联网平台校验应用服务器合法性。在调测时，您可以通过<a href="#">制作自签名证书</a>进行调测。在商用时，建议您向证书知名机构申请和购买商用证书，以确保证书的安全性。</li> </ul>
接入设备管理服务	<p>物联网平台提供了<a href="#">开发中心（调测平台）</a>和<a href="#">商用管理平台</a>，您在进行应用开发和调试时，可以先接入开发调测平台，待功能开发完善，具备商用使用条件时，再接入商用管理平台。</p> <p>应用接入地址获取请参考<a href="#">平台对接信息</a>，应用ID和应用密钥为在商用管理平台上“创建应用”时分配。</p>

表 3-1 开发方式对比说明

开发方式	优点	缺点	适用场景
<a href="#">调用API接口</a>	<ul style="list-style-type: none"> <li>开发灵活，按需调用API接口。</li> <li>对于应用开发语言无限制，支持所有的开发语言</li> </ul>	<ul style="list-style-type: none"> <li>开发工作量、开发难度相比集成SDK大。</li> <li>应用上线周期相对较长。</li> </ul>	<ul style="list-style-type: none"> <li>企业开发能力强，需灵活使用物联网平台的能力。</li> <li>企业已有应用服务器，需要对接物联网平台。</li> </ul>
<a href="#">集成SDK</a>	<ul style="list-style-type: none"> <li>代码开发工作量较小，开发能力的门槛相比直接调用API接口较低。</li> <li>开发周期短，可以快速构建应用服务器。</li> </ul>	<ul style="list-style-type: none"> <li>与直接调用API接口相比，开发的灵活性稍差。</li> <li>开发语言支持Java、PHP和Python，暂还不支持所有的开发语言。</li> </ul>	适用于企业对应用的个性化的定制要求不高，能够快速构建和上线应用。

# 4 平台侧开发

- 4.1 体验开发中心（联通用户专用）
- 4.2 创建项目（联通用户专用）
- 4.3 创建产品（联通用户专用）
- 4.4 开发Profile
- 4.5 开发编解码插件
- 4.6 调测产品（联通用户专用）

## 4.1 体验开发中心（联通用户专用）

非联通用户请查看[设备接入服务](#)。

本文通过“智慧路灯”为示例，通过开发中心提供的设备模拟器和应用模拟器替代真实的设备和应用，带您快速体验设备上报数据到物联网平台和远程下发控制命令到设备的全过程。

假设：

路灯设备上报一条数据消息，包含路灯的光照强度（Light\_Intensity）和路灯的开关状态（Light\_Status）；支持远程控制路灯开关状态的命令（SWITCH\_LIGHT），上报数据的格式为二进制格式。

### 前提条件

已经注册华为官方帐号，具体可参考[华为账号注册](#)。

已经在华为云上完成实名制认证，具体可参考[实名认证](#)。

### 开发中心申请和创建项目

用户第一次使用物联网平台云服务时，首先需要申请开通“开发中心”服务。基于开发中心，用户可以在线开发设备的Profile文件和编解码插件；另外开发中心提供了设备和应用模拟器，可以便捷地调测开发的Profile文件和编解码插件的正确性。

**步骤1** 登录华为云官方网站，访问IoT设备管理服务。

- 步骤2** 在设备管理云服务首页，点击“**开发中心**”开通业务。
- 步骤3** 填写开通信息，本示例以选择“智能路灯解决方案”为例，点击“**立即开通**”。
- 步骤4** （可选）进入“开发中心”首页后，如果用户是第一次使用，首先需要点击右上角的“**厂商信息**”，编辑并完善厂商信息后，返回主页。
- 步骤5** 在“开发中心”首页，点击“**新建项目**”，创建一个新的物联网产品项目，填写项目名称、所属行业后，点击“**确定**”。

The screenshot shows a modal dialog titled "新建项目" (New Project). It includes a close button (X) in the top right corner. The form contains the following fields:

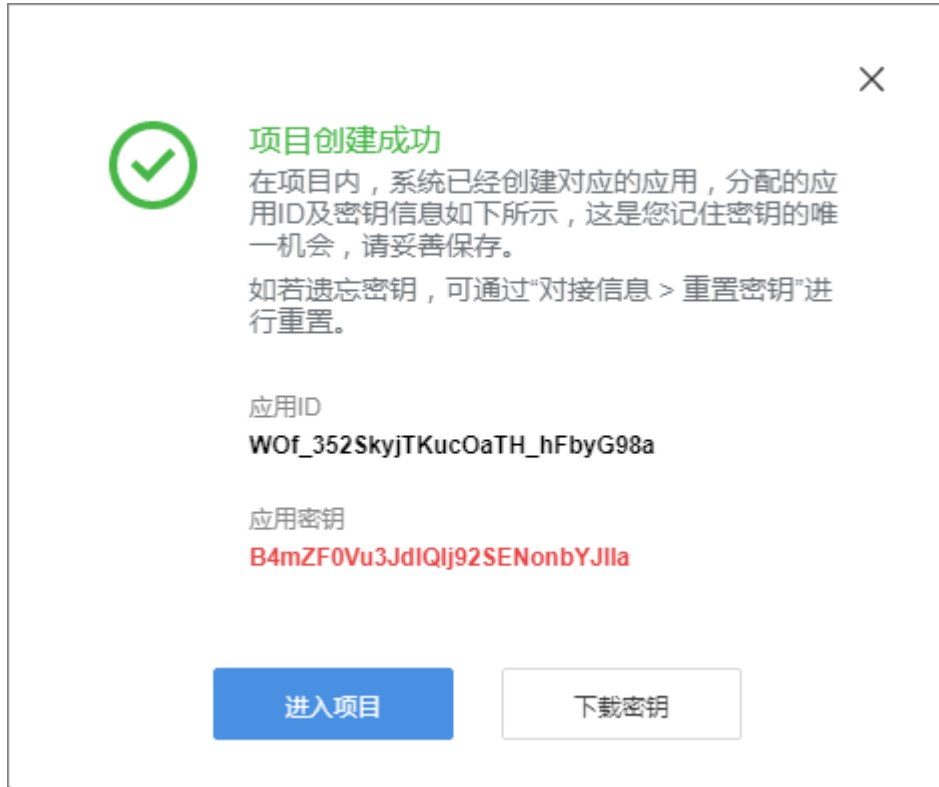
- \*项目名称** (Project Name): A text input field containing "StreetLight001".
- \*所属行业** (Industry): A dropdown menu with "公用事业 (NB-IoT)" selected.
- 描述** (Description): A large empty text area.

A blue "确定" (Confirm) button is located at the bottom center of the dialog.

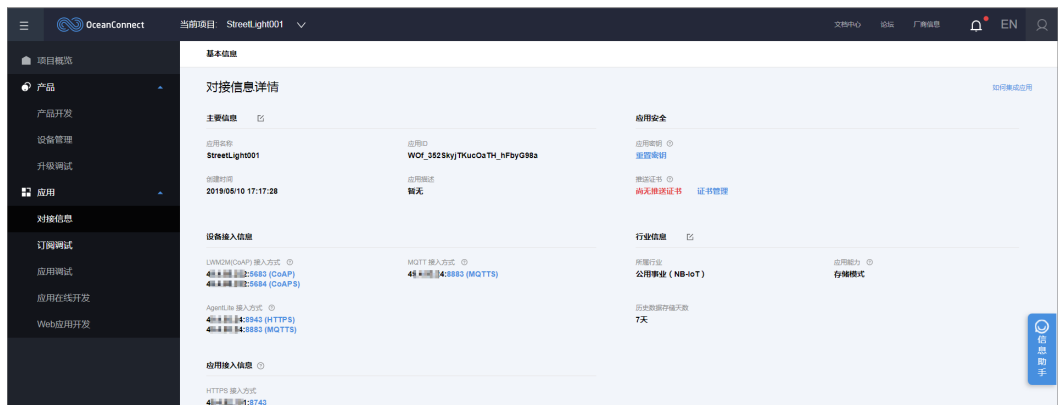
- 步骤6** 创建完项目后，会生成应用ID和应用密钥，请将密钥下载到本地并妥善保管，然后进入项目。

#### 📖 说明

应用密钥用于应用服务器接入的鉴权，在界面上不可见，请妥善保管，如果忘记了密钥，可以在“**对接信息**”功能中，对密钥进行重置。



**步骤7** 项目创建完成后，可以在“**对接信息**”中，查看项目的基本信息，例如设备接入信息、应用接入信息、重置密钥等。

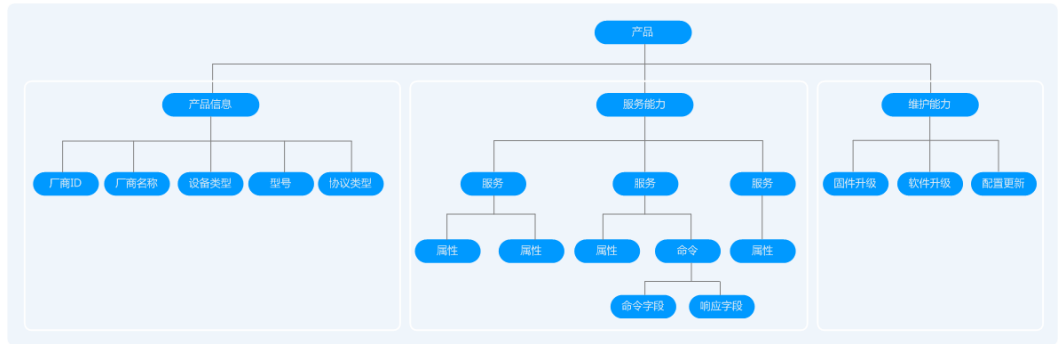


----结束

## 产品在线开发

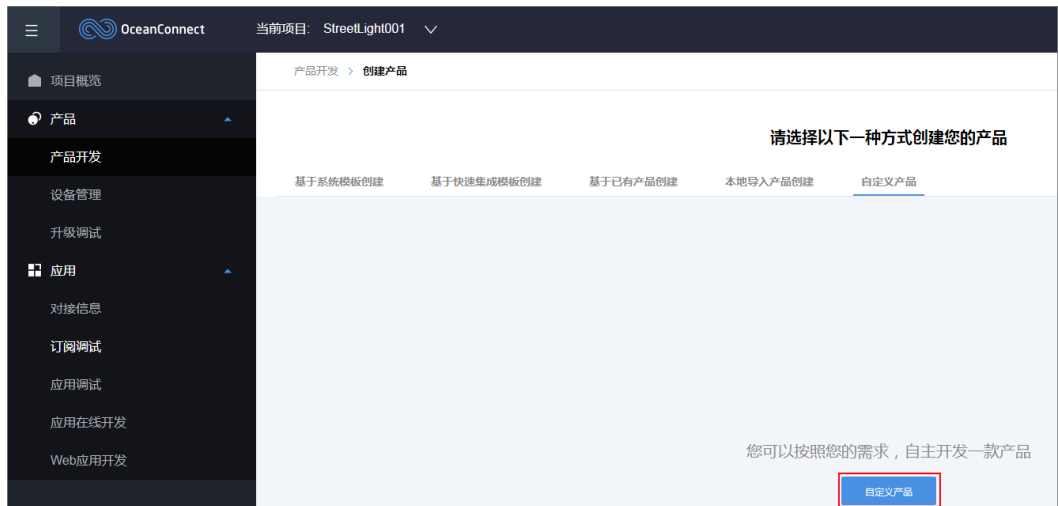
产品模型（也称Profile）用于描述设备具备的能力和特性。开发者通过定义Profile文件，在物联网平台构建一款设备的抽象模型，使平台理解该款设备支持的服务、属性、命令等信息，例如温度、光强度、开关等。





本示例以一款智慧路灯为例进行描述，假设路灯设备上报一条数据消息，包含路灯的光照强度（Light\_Intensity）和路灯的开关状态（Light\_Status）；支持远程控制路灯开关状态的命令（SWITCH\_LIGHT）。

**步骤1** 在“产品开发”界面，点击“新建产品”，然后选择“自定义产品”。



**步骤2** 在弹出的对话框中，设置产品信息，完成后点击“创建”。

### 设置产品信息 ? ×

\* 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 ?

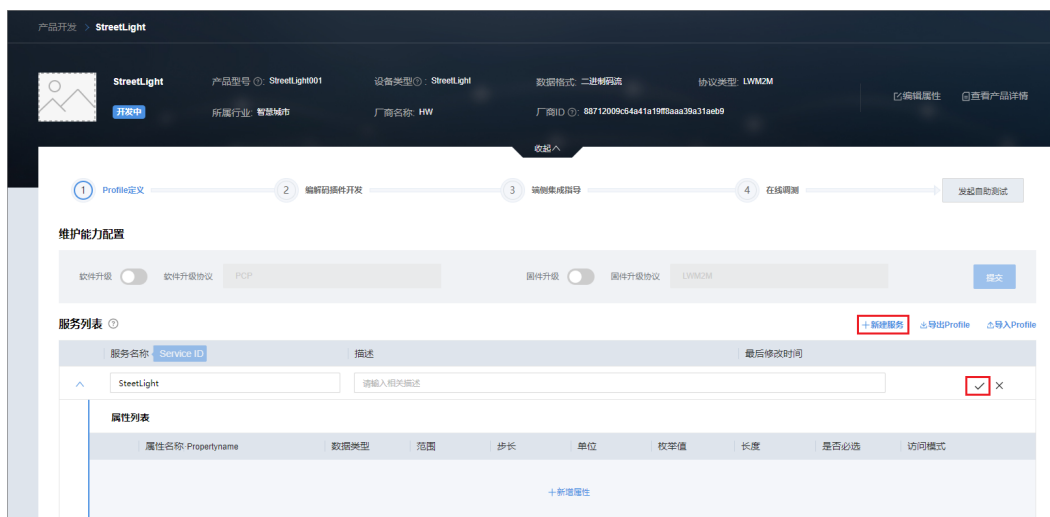
注意：LWM2M协议的设备需要完善数据解析，将设备上报的二进制数据转换为平台上的JSON数据格式

\* 数据格式：

产品图片： 



**步骤3** 进入到Profile定义界面后，点击“新建服务”，填写服务信息，然后点击“√”。



**步骤4** 定义路灯上报的环境光强度和路灯开关状态的属性。首先输入“服务名称”：StreetLight，然后点击“新增属性”。

**步骤5** 首先定义一条属性为：路灯采集的当前环境的光照强度，数据类型为int，光照强度范围为：0~100。

### 新增属性 ×

\* 名称

\* 数据类型

\* 最小值  \* 最大值

步长  单位

\* 访问模式  
 R 属性值可读  
 W 属性值可写 (更改)  
 E 属性值更改时上报事件

是否必选  
 是

**步骤6** 再点击“添加属性”按钮，定义一条属性为：路灯当前的开关灯状态，数据类型为 int，0代表关闭，1代表打开状态。

**新增属性**
✕

**\* 名称**

**\* 数据类型**

int ▼

**\* 最小值**

**\* 最大值**

步长

单位

**\* 访问模式**

R 属性值可读

W 属性值可写（更改）

E 属性值更改时上报事件

是否必选

是

确定

取消

**步骤7** 接下来定义远程控制开关灯状态的命令。点击“**添加命令**”按钮，定义命令名称为：SWITCH\_LIGHT。

**步骤8** 点击“**添加下发命令字段**”，命令名称为：SWITCH\_LIGHT，数据类型为：string，长度为：3个字符，枚举值为：ON,OFF。

### 新增下发命令字段 ×

\* 名称

\* 数据类型

\* 长度

枚举值 (值之间以英文逗号分隔)

是否必选

是

**步骤9** 点击“确定”按钮，完成该路灯的Profile文件创建。

----结束

## 编解码插件在线开发

通常情况下设备为了省电，设备会采用“二进制”格式上报数据，编解码插件的作用就是将设备上报的“二进制”格式数据，按照Profile文件的定义的属性转换为“JSON”格式数据，便于物联网平台和应用服务器识别。同时，用户远程下发控制命令时，物联网平台会将“JSON”格式的命令转换为“二进制”格式数据下发给设备。

**注：**如果设备本身上报的是JSON格式数据，则不需要定义编解码插件，如使用Agent Lite接入的设备。

**步骤1** 在“产品开发”页面进入到创建的产品中，选择“编解码插件开发”。

**步骤2** 在“在线编解码插件编辑器”中，点击“新增消息”。



**步骤3** 输入消息名：LightData，消息类型选择：数据上报，然后再点击“添加字段”。



**步骤4** 输入上报消息的名字：LightIntensity，数据类型：int8u（8位无符号整型），长度：1字节，单击“完成”。

### 添加字段 ×

标记为地址域 ?

**\*名字**

LightIntensity

**描述**

解析上报光照强度

**数据类型 (大端模式)**

int8u(8位无符号整型) ▼

**\* 长度 ?**

1

**默认值 ?**

默认值

**偏移值 ?**

0-1

完成 取消

**步骤5** 再次单击“添加字段”，添加路灯上报的路灯开关状态数据。

**步骤6** 输入名字：LightStatus，数据类型：int8u（8位无符号整型），长度：1字节，单击“完成”。



**添加字段**
✕

标记为地址域 ?

**\*名字**

描述

描述

数据类型 (大端模式)

int8u(8位无符号整型) ▼

**\* 长度** ?

默认值 ?

偏移值 ?

完成

取消

**步骤7** 单击“完成”，完成路灯上报数据的编解码定义。

**步骤8** 再次单击“新增消息”，定义远程控制路灯开关的命令对应的编解码插件消息。

**步骤9** 输入消息名：SwitchStatus，消息类型选择：命令下发，然后再点击“添加字段”。

新增消息

基本信息

\*消息名  
SwitchStatus

消息描述  
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

**步骤10** 输入名字：SwitchStatus，数据类型：string（字符串类型），长度：3字符，单击“完成”。

### 添加字段 ×

标记为地址域 ?

标记为响应标识字段 ?

**\*名字**

SwitchStatus

**描述**

远程控制路灯的开关状态

**数据类型 (大端模式)**

string(字符串类型) ▼

**\* 长度 ?**

3

**默认值 ?**

默认值

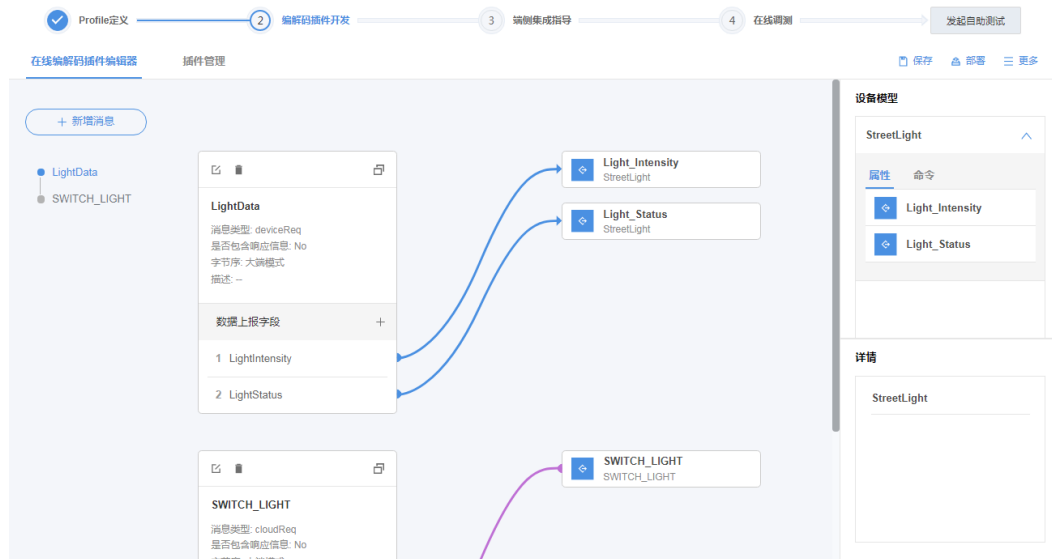
**偏移值 ?**

0-3

完成 取消

**步骤11** 单击“完成”，完成远程控制命令的编解码定义。

**步骤12** 拖动右侧“设备模型”区域的属性字段和命令字段（Profile文件定义的字段），与编解码插件定义的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13** 编解码插件与Profile文件建立映射完成后，单击右上角“保存”，并单击“部署”，完成编解码插件的部署。

----结束

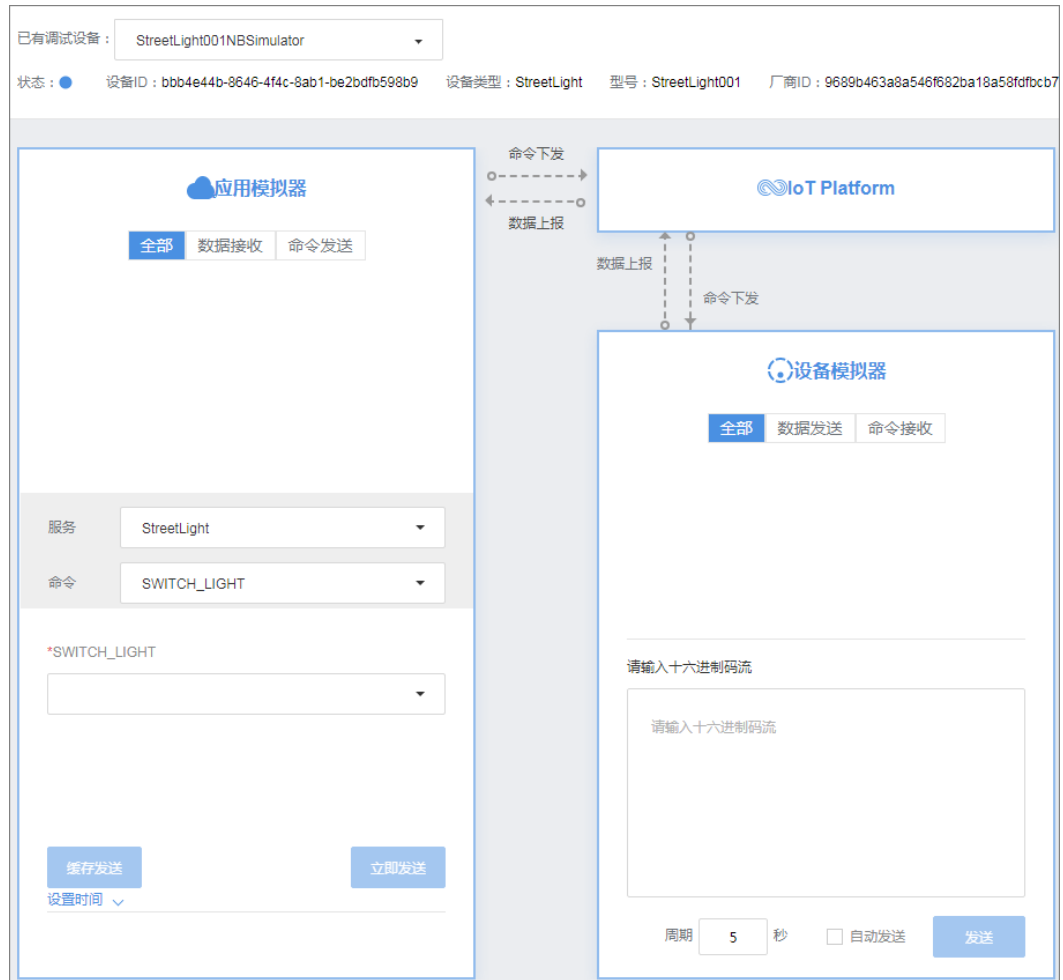
## 使用在线模拟器调试

模拟器在线调测具备设备模拟和应用模拟功能，可以对定义的Profile文件和编解码插件进行调试，用户可以直观的感受设备上报数据到物联网平台，以及使用物联网平台下发远程控制命令。

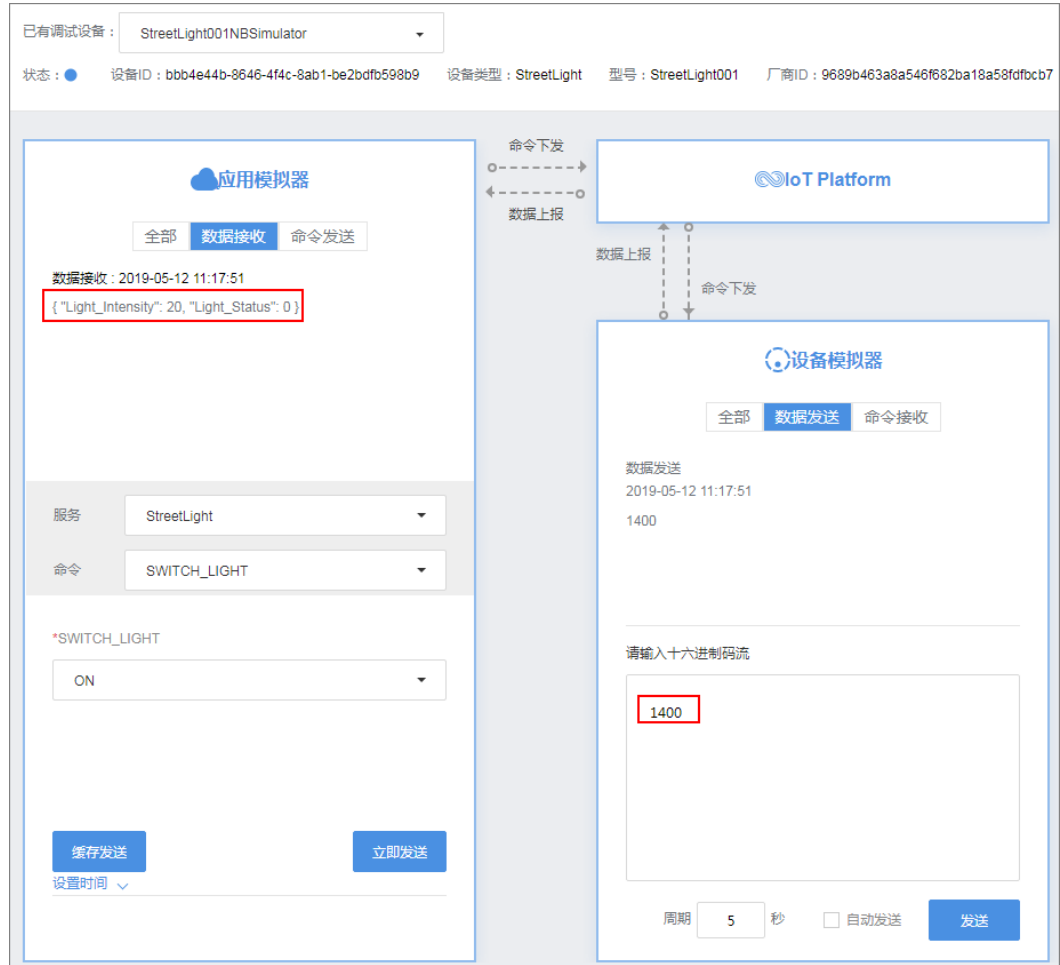
**步骤1** 在“产品开发”页面进入到创建的产品中，选择“在线调测”，并单击“新增测试设备”。



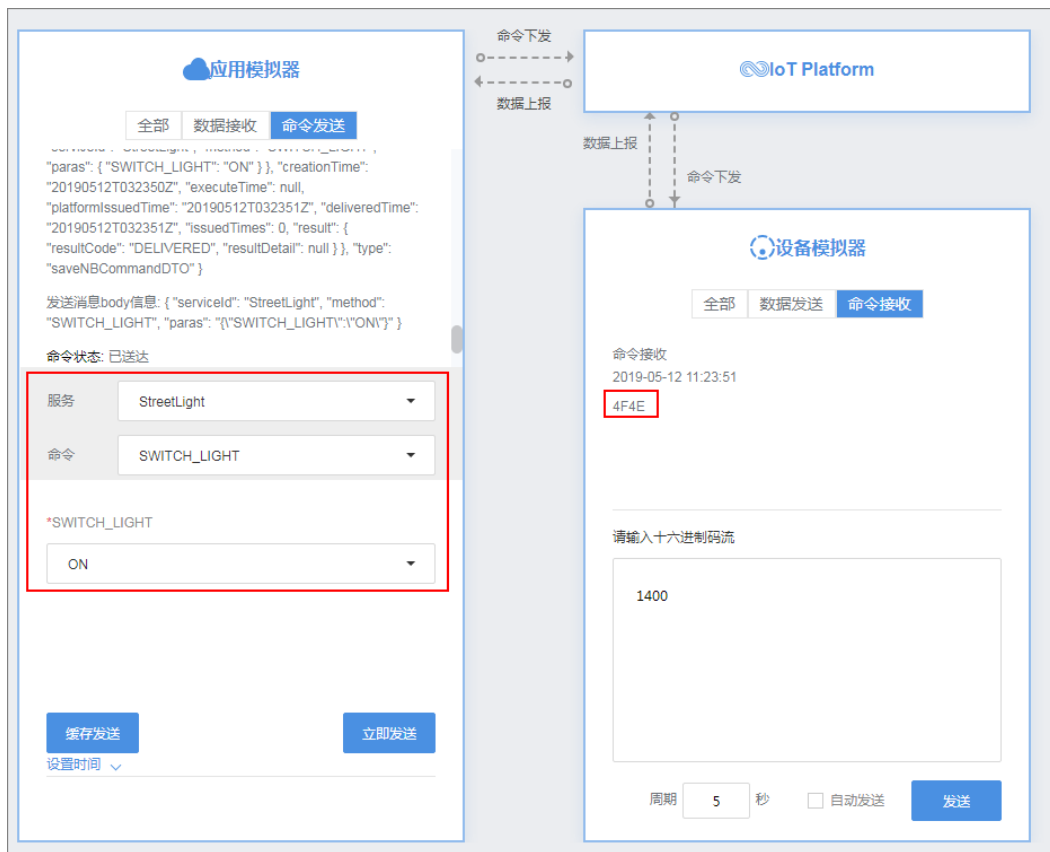
**步骤2** 选择“没有真实的物理设备”，使用模拟器进行调测，单击“创建”后，系统会默认创建一个模拟设备，并接入调测界面。



**步骤3** 模拟设备数据上报场景，假设上报路灯采集的光照强度为：20，路灯开关状态为：0（关闭），则在设备模拟器中，输入十六进制码流：1400（光照强度消息为第一个字节，对应的十六进制码流为14；路灯开关状态为第二字节，对应的十六进制码流为00），然后单击“发送”，我们可以在应用模拟器中看到转换为JSON格式的数据为：“Light\_Intensity”: 20, "Light\_Status": 0。



**步骤4** 模拟远程下发控制命令场景，在应用模拟器中，选择服务：StreetLight，命令：SWITCH\_LIGHT，命令取值为：ON，单击“立即发送”，我们可以在设备模拟器中看到转换为十六进制的码流：4F4E（经ASCLL码转换为十六进制）。

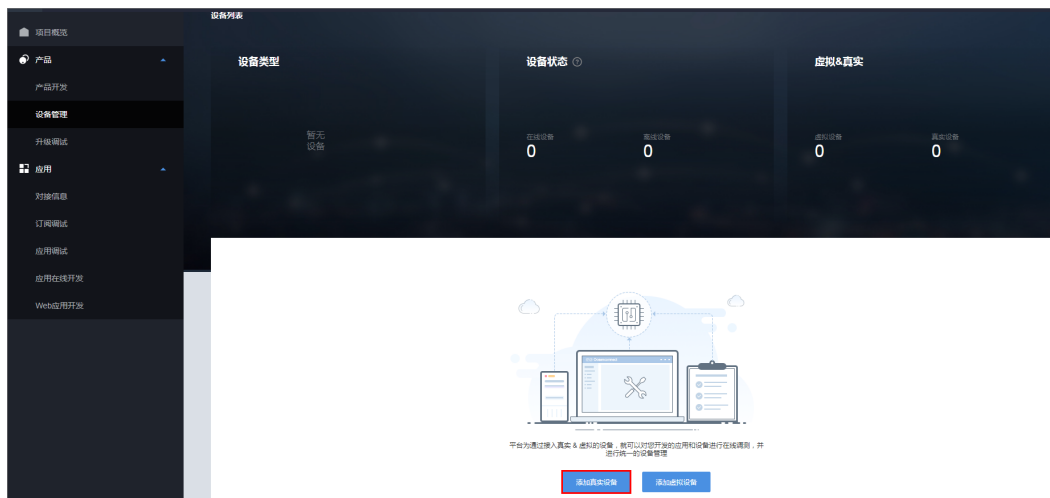


----结束

## 使用离线模拟器调试

**NB-IoT设备模拟器**用于模拟以CoAP/LWM2M协议接入物联网平台的NB设备，实现数据上报和命令下发功能。

**步骤1** 选择“产品 > 设备管理”，单击“添加真实设备”。



**步骤2** 选择前面创建的产品，并填写设备相关信息，单击“确定”。

注：如果使用DTLS传输层安全协议接入时，设备注册方式选择“加密”，且请妥善保存PSK码。

如下信息根据实际情况填写：

- 设备名称：NBdevice01
- 设备标识：aaaaa11111
- 安全密钥(PSK)：aaaaa11111aaaaa
- 设备注册方式：加密

设备创建成功后，将返回“设备名称”、“设备ID”和“PSK码”。

**步骤3** 下载并解压NB-IoT设备模拟器，然后双击“NB-IoTDeviceSimulator\_zh.jar”，运行模拟器。

注：请确保已经安装jdk，否则，无法运行jar文件。

images	2017/11/2 15:24	文件夹	
Californium.properties	2017/7/29 12:39	PROPERTIES 文件	2 KB
NB-IoTDeviceSimulator_en.jar	2017/8/7 14:08	Executable Jar File	4,361 KB
<b>NB-IoTDeviceSimulator_zh.jar</b>	2017/8/7 14:07	Executable Jar File	4,361 KB
setting.properties	2017/8/7 15:21	PROPERTIES 文件	1 KB

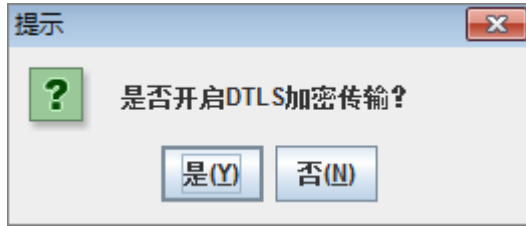
NB-IoT设备模拟器的文件说明如下：

- **Californium.properties**为模拟器的配置文件。
- **NB-IoTDeviceSimulator\_en.jar**为英文版模拟器。
- **NB-IoTDeviceSimulator\_zh.jar**为中文版模拟器。
- **setting.properties**为设备模拟器接入物联网平台的配置文件。

**步骤4** 模拟器启动后，会提示“是否开启DTLS加密传输？”，单击“是”。

注：如果模拟设备不使用DTLS协议接入物联网平台，单击“否”。





**步骤5** 填写“IP地址”、“VerifyCode”和“PSK”，单击“注册设备”，将模拟器与物联网平台进行绑定。

**注：**如果未开启DTLS加密传输，则无需填写PSK。

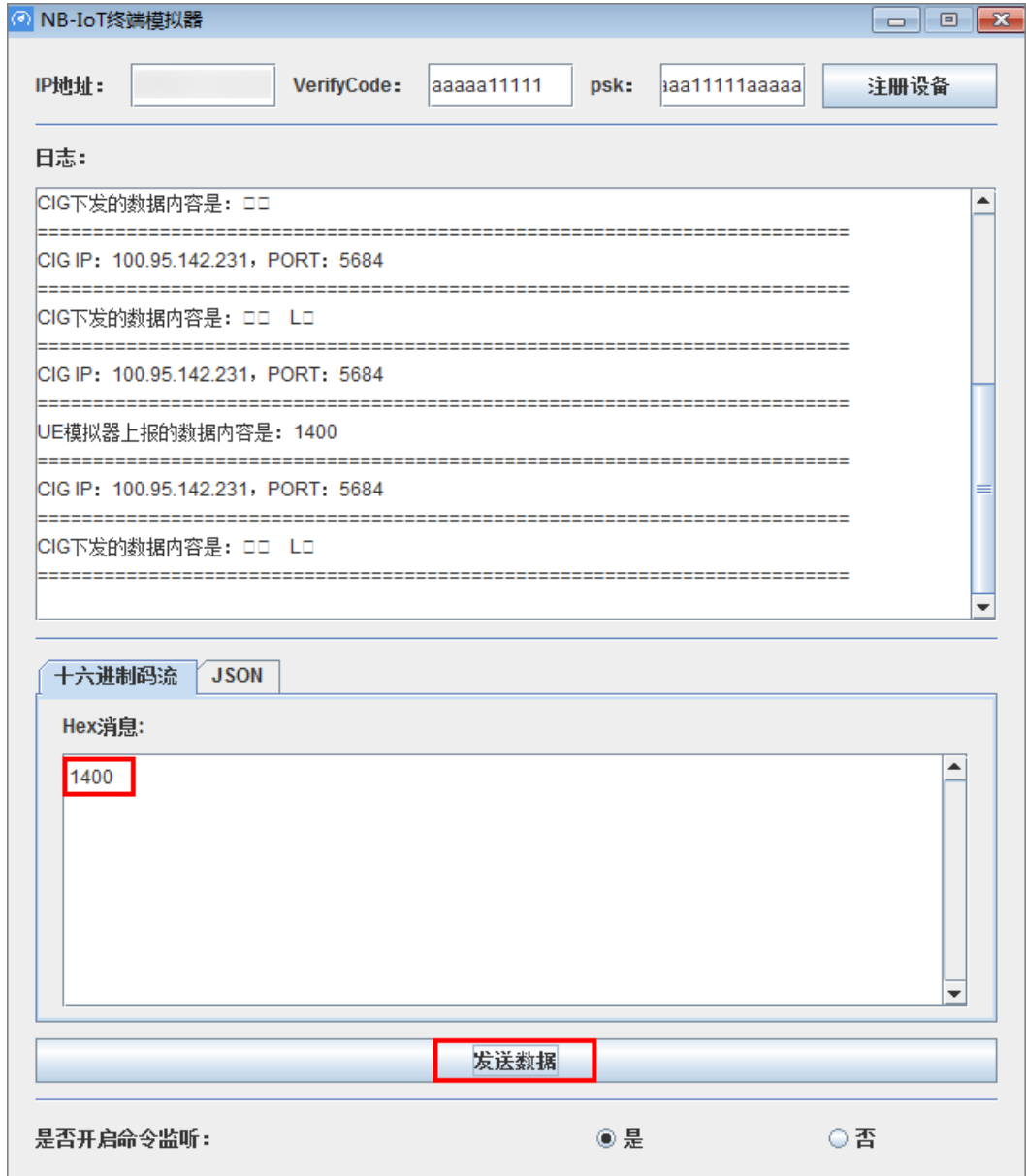
如下信息根据实际情况填写：

- IP地址：物联网平台的接入地址，请到开发中心“应用 > 对接信息”查看。
- VerifyCode：设备标识码，如：aaaaa11111。
- PSK：aaaaa11111aaaaa



设备绑定成功，可以回到开发中心，选择“产品 > 设备管理”，看见设备“aaaaa11111”显示“在线”，表示模拟器绑定成功。

**步骤6** 模拟设备数据上报场景，假设上报路灯采集的光照强度为：20，路灯开关状态为：0（关闭），则在NB-IoT设备模拟器中，输入十六进制码流：1400（光照强度消息为第一个字节，对应的十六进制码流为14；路灯开关状态为第二字节，对应的十六进制码流为00），然后单击“发送数据”。

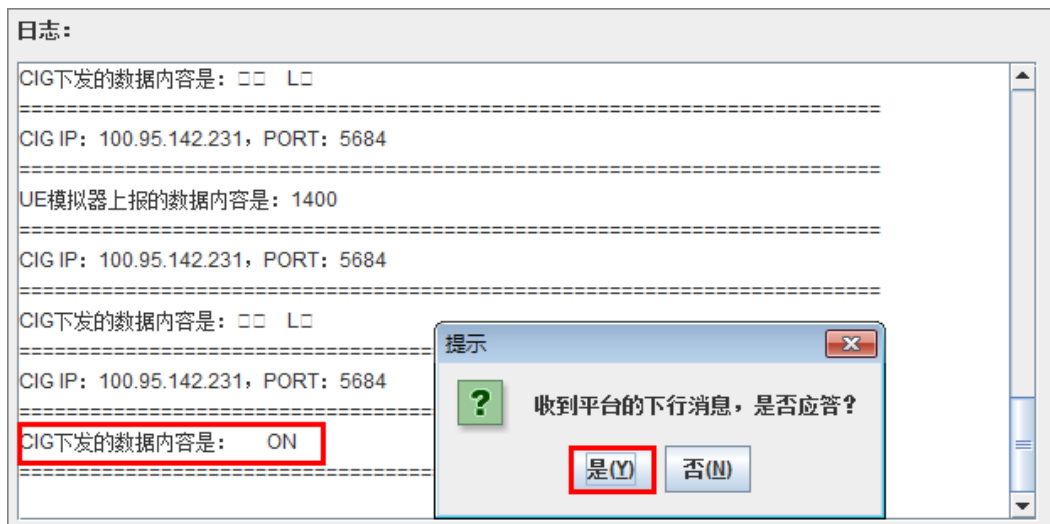


数据上报成功，可以回到开发中心，在设备“aaaaa11111”的“历史数据”页面中查看到转换为JSON格式的数据为：“Light\_Intensity”: 20, “Light\_Status”: 0。

**步骤7** 模拟远程下发控制命令场景，在开发中心，选择“设备管理”，单击设备“aaaaa11111”右边的“调试产品”，进入调试界面。

在应用模拟器中，选择服务：StreetLight，命令：SWITCH\_LIGHT，命令取值为：ON，单击“立即发送”。

我们可以在“日志”栏看到CIG下发的数据内容：ON，且模拟器会提示“收到平台的下行消息，是否应答？”，单击“是”，可以在开发中心的应用模拟器看到命令状态为“已送达”。



----结束

## 进阶体验

按照本页面的指导完成开发中心快速体验, 您应该已经基本了解了设备管理服务的在线开发流程, 包括创建项目, 开发产品模型、编解码插件和使用模拟器在线调测。

若您想要进一步体验设备管理服务, 可参考[开发指南](#)完成真实应用和设备的对接, 使用真实应用或控制台体验更多功能, 控制台的操作指导请参见[使用指南](#)。

## 4.2 创建项目（联通用户专用）

非联通用户请查看[设备接入服务](#)。

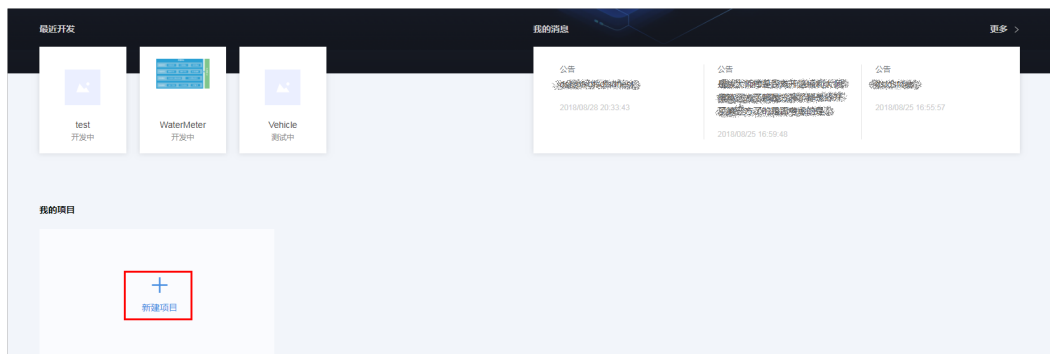
项目是物联网平台提供物联网应用和设备的调测空间, 您可以根据场景的不同创建不同项目空间分别调测。

- 创建项目时, 物联网平台会分配一个**应用ID**（接口调用时参数名为**appld**）作为项目的唯一标识。若应用服务器需要调用物联网平台的API接口进行一些业务处理, 请求中必须携带**appld**以接入对应的项目空间, 鉴权接口携带在Body中, 其他接口的**appld**的值一般携带在Header中。
- 创建项目后, 可以在项目中查看应用服务器和设备的接入地址和端口信息, 方便您快速对接应用服务器和设备。
- 项目被删除后项目内的所有资源, 如设备、产品、订阅数据在平台中的信息会被全部删除, 并且不可恢复, 请谨慎操作。

### 创建项目

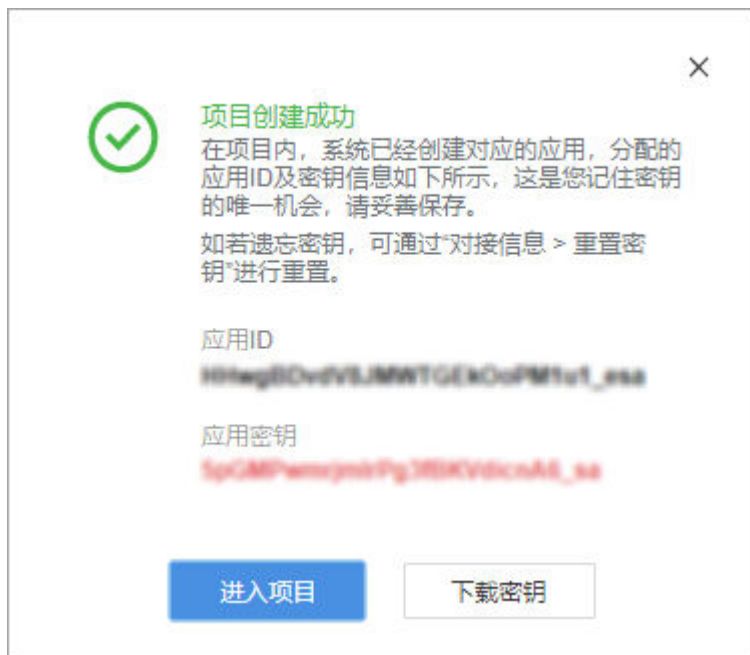
在基于开发中心进行物联网开发时, 您需要根据行业属性创建独立的项目, 并在该项目空间内开发物联网产品和应用。

**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”。“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。最多支持创建10个项目。

**步骤3** 项目创建成功后，系统返回“应用ID”和“应用密钥”。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。



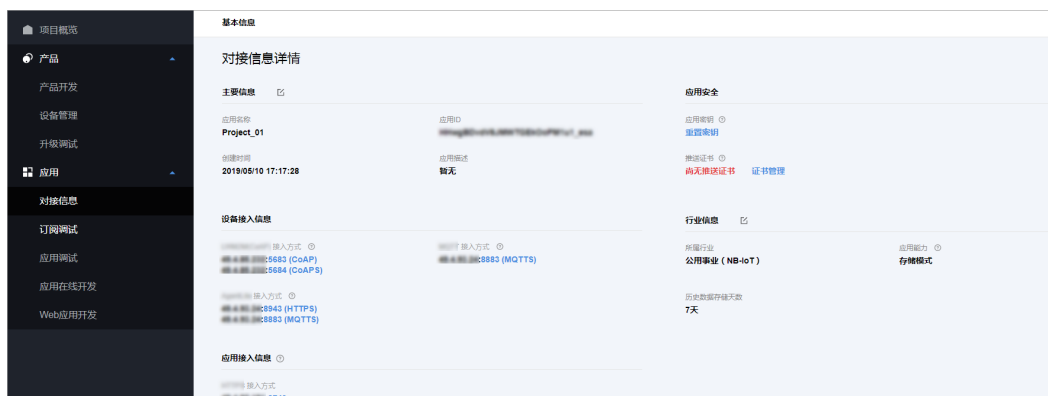
**步骤4** 选择新创建的项目，可以进入项目空间。在项目空间内，您可以快速完成Profile文件和编解码插件的在线开发和调试，获取设备和应用的开发帮助，以及通过模拟器完成设备或应用的对接调试。



----结束

## 查看项目信息

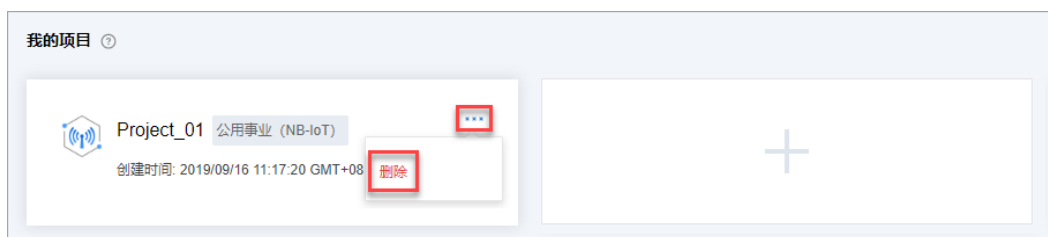
项目创建完成后，您可以在“**对接信息**”中，查看项目的基本信息，例如**设备接入地址**、**应用接入地址**、重置密钥等。



## 删除项目

项目被删除后项目内的所有资源，如设备、产品、订阅数据在平台中的信息会被全部删除，并且不可恢复，请谨慎操作。

**步骤1** 在开发中心首页的“我的项目”中，选择需要删除的项目，点击项目右上角的“...”按钮，然后点击“删除”。



**步骤2** 系统将弹出“删除项目确认”窗口，点击“确认”完成项目删除。



----结束

## 4.3 创建产品（联通用户专用）

非联通用户请查看[设备接入服务](#)。

在物联网平台中，某一类具有相同能力或特征的设备的合集被称为一款产品。产品包含Profile（产品模型）、编解码插件、测试报告等资源，其中产品信息被记录在Profile中。

- 在一个项目里最多可以创建20个产品。

- 产品开发最重要的是开发Profile，Profile用于描述设备具备的能力和特性。定义Profile，即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。如果设备上报的数据是二进制码流格式，就需要开发对应的编解码插件，用于物联网平台完成二进制格式和JSON格式的转换。
- 删除产品后，该产品下的Profile、编解码插件等资源将被清空，请谨慎操作。

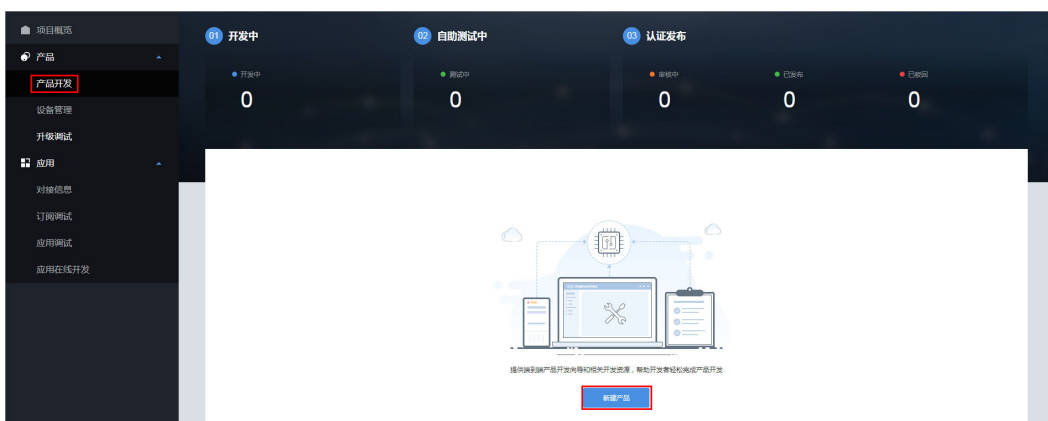
## 创建产品

开发中心上提供了多种创建产品的方法。

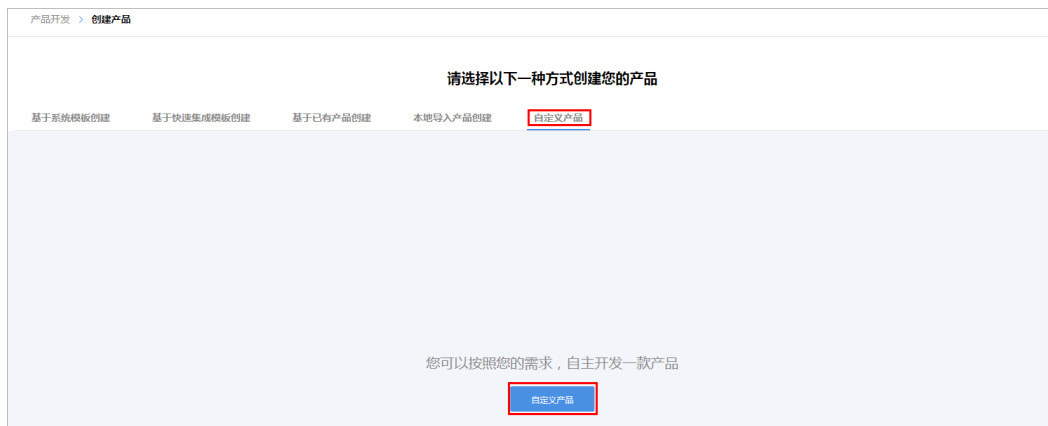
方式	描述
基于系统模板创建	开发中心提供了多种产品模板，这些模板涉及多个领域。模板中提供了已经编辑好的Profile文件，您可以根据需要对Profile中的字段进行修改和增删。有的模板同时提供了开发好的编解码插件，用户也可以进行修改。
基于快速集成模板创建	类似系统模板，快速集成也是提供了一些产品模板，这些模板适用快速体验平台的集成对接，或者适用一些研讨或培训的学习场合。
基于已有产品创建	您可以基于已有的产品进行克隆，创建新的产品。
本地导入产品创建	将本地写好的Profile文件上传到平台，开发一个新产品。
自定义产品	您可以从零自定义构建产品。

接下来以创建自定义产品为例，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”等信息后，点击“创建”。

- “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”、“接入应用层协议类型”和“数据格式”等信息根据实际情况进行填写。
- 当“数据格式”配置为“二进制码流”时，该产品下需要进行编解码插件开发；当“数据格式”配置为“JSON”时，该产品下不需要进行编解码插件开发。



### 设置产品信息 ? ×

\* 产品名称:

\* 产品型号:

\* 厂商ID:


\* 所属行业:


\* 设备类型:

\* 接入应用层协议类型 ?

注意: LWM2M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:

产品图片:  



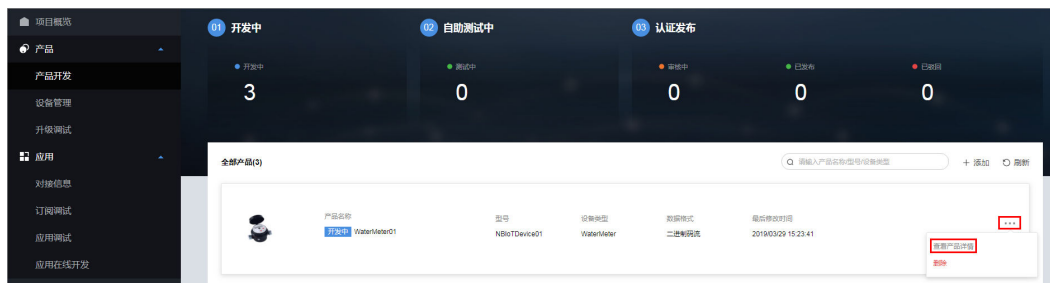
**步骤4** 在“产品开发”界面将会呈现已经创建的产品，选择具体产品，可以进入该产品的开发空间。



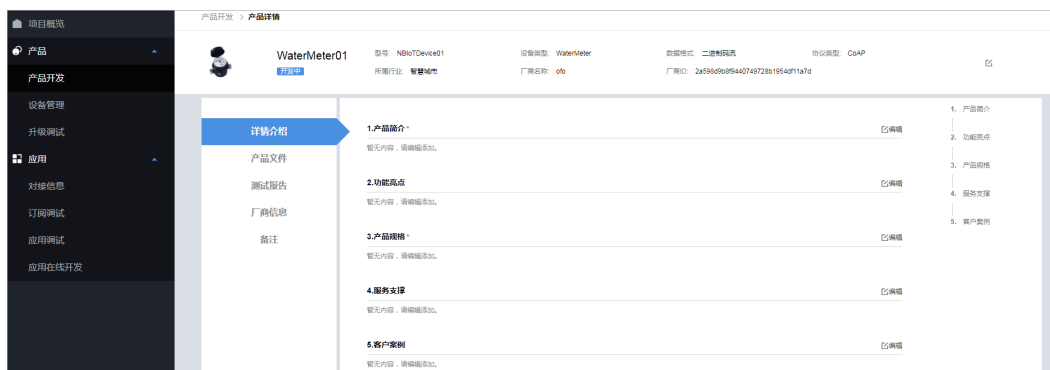
----结束

## 完善产品详情

**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”界面，可以查看产品的“详情介绍”、“产品文件”、“测试报告”、“厂商信息”等。在“详情介绍”界面，可以对“产品简介”、“功能亮点”、“产品规格”等信息进行完善。



----结束

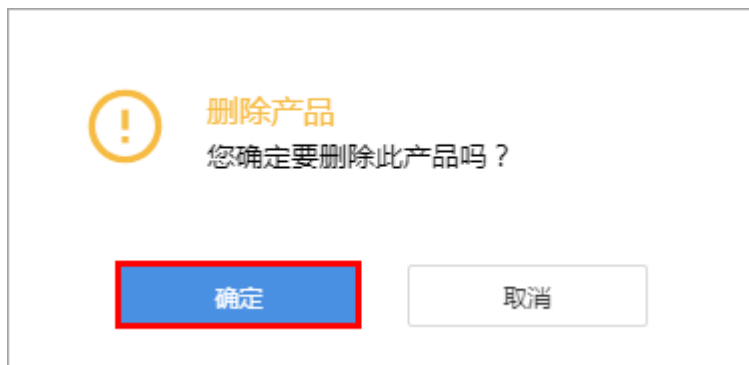
## 删除产品

删除产品后，该产品下的Profile、编解码插件等资源将被清空，请谨慎操作。

**步骤1** 在“产品开发”界面选择需要删除的产品，点击该产品右侧“...”按钮，然后点击“删除”。



**步骤2** 系统将弹出删除产品确认窗口，点击“确定”完成产品删除。



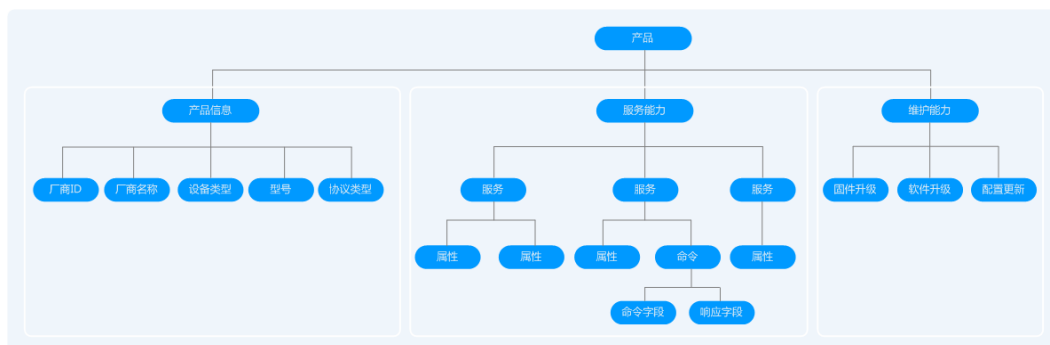
---结束

## 4.4 开发 Profile

### 4.4.1 什么是 Profile（联通用户专用）

非联通用户请查看[设备接入服务](#)。

Profile（即产品模型）是用来描述一款产品中的设备“是什么”、“能做什么”以及“如何控制该设备”的文件。在物联网平台集成对接中需要先创建Profile，因为Profile里面定义了设备上报的数据和应用服务器下发的命令包含了哪些字段。定义Profile，即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、升级能力等信息。



Profile主要包含产品信息、服务能力和维护能力三部分。

- **产品信息**

描述一款水表设备的基本信息，包括厂商ID、厂商名称、产品类型、型号、协议类型。其中厂商ID和型号唯一标识一款产品。

例如，某个水表的制造厂商为“HZYB”，制造商ID为“TestUtf8Manuld”，型号为“NBloTDevice”，协议类型为“CoAP”。

- **服务能力**

服务能力用于描述设备具备的业务能力。将设备业务能力拆分成若干个服务后，再定义每个服务具备的属性、命令以及命令的参数。

以水表为例，水表具有多种能力，如上报水流、告警、电量、连接等各种数据，并且能够接受服务器下发的各种命令。Profile文件在描述水表的能力时，可以将水表的能力划分五个服务，每个服务都需要定义各自的上报属性或命令。说明如下：

服务名	描述
基础 (WaterMeterBasic)	用于定义水表上报的水流量、水温、水压等参数，如果需要命令控制或修改这些参数，还需要定义命令的参数。
告警 (WaterMeterAlarm)	用于定义水表需要上报的各种告警场景的数据，必要的话需要定义命令。
电池 (Battery)	定义水表的电压、电流强度等数据。
传输规则 (DeliverySchedule)	定义水表的一些传输规则，必要的话需要定义命令。
连接 (Connectivity)	定义水表连接参数。

注：具体定义几个服务是非常灵活的，如上面的例子可以将告警服务拆分成水压告警服务和流量告警服务，也可以将告警服务合入到水表基础服务中。

- **维护能力**

描述设备具备软固件升级、配置更新等能力。

## 在线开发和离线开发

Profile的开发手段有在线开发和离线开发两种，我们推荐使用开发中心**在线开发Profile**。

- **在线开发**即在开发中心上，通过界面操作开发Profile，开发完成后可以下载。
- **离线开发**是指通过了解Profile格式规范，在本地进行开发、打包并上传。

### 4.4.2 在线开发 Profile（联通用户专用）

非联通用户请查看[设备接入服务](#)。

在线创建Profile前需要先[创建项目](#)并[定义产品](#)。创建产品需要输入manufactureId、manufactureName、deviceType、Model等信息，Profile会使用这些信息作为设备能力字段取值。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，您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。

本节定义包含一个服务的Profile为示例，该Profile包含设备上报数据、下发命令、下发命令响应、软固件升级等场景的服务和字段。

**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“添加属性”，在弹出窗口中配置属性的各项参数，点击“确定”。

- 名称：首位必须为字母，建议采用驼峰形式，如batteryLevel、internalTemperature。

- 数据类型：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。配置“经纬度”属性时, 数据类型建议使用“decimal”。
  - **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。
- 访问模式: 设置应用服务器通过接口访问数据的模式:
- R: 通过接口可以查询该属性。
  - W: 通过接口可以修改该属性值。
  - E: 应用服务器订阅了数据变化通知后, 设备上报了属性, 应用服务器会收到推送通知。
- 是否必选: 设备上报的这个属性是不是必选。

**新增属性**

\* 名称  
batteryLevel

\* 数据类型  
int

\* 最小值: 0      \* 最大值: 100

步长: 1      单位:

\* 访问模式  
 R 属性值可读  
 W 属性值可写 (更改)  
 E 属性值更改时上报事件

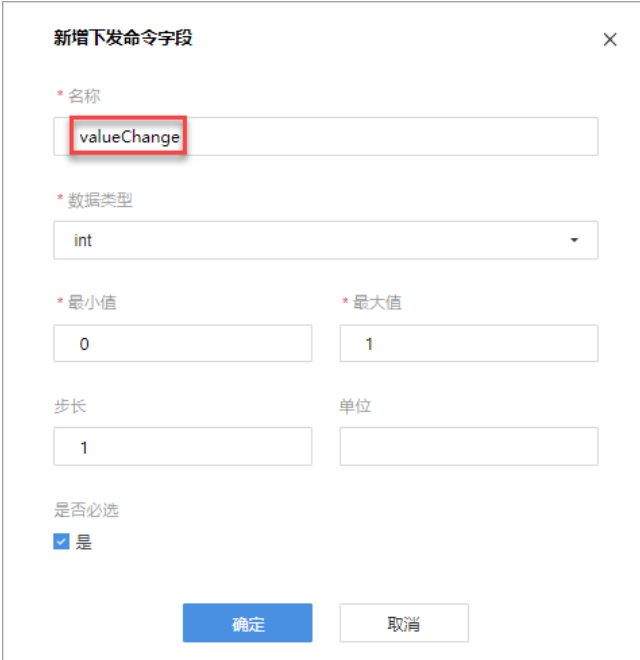
是否必选  
 是

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”首位必须为字母, 建议采用全大写形式, 单词间用下划线连接的命名方式, 如DISCOVERY, CHANGE\_STATUS。



4. 点击“添加下发命令字段”，在弹出窗口中配置下发命令字段的各项参数，点击“确定”。“下发命令字段名称”首位必须为字母，建议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valueChange；其余参数，请根据此类设备的实际情况进行配置。



5. 如果要添加命令响应，点击“添加响应命令字段”，在弹出窗口中配置响应命令字段的各项参数，点击“确定”。“响应命令字段名称”首位必须为字母，建议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valueResult；其余参数，请根据此类设备的实际情况进行配置。

**新增响应命令字段**

\* 名称  
valueResult

\* 数据类型  
int

\* 最小值  
0

\* 最大值  
1

步长  
1

单位

是否必选  
 是

确定 取消

**步骤4** 如果要添加软件/固件升级能力，在“维护能力配置”下，开启“软件升级”/“固件升级”。

维护能力配置

软件升级 软件升级协议 PCP

固件升级 固件升级协议 LWM2M

提交

提交后，Profile中会自动多出一个“DM服务”应用于升级。

服务列表

服务名称	Service ID	描述	最后修改时间
DM			--

属性列表

属性名称	Property Name	数据类型	范围	步长	单位	枚举值	长度	是否必选	访问模式
swVersion		string	--	--	--	--	256	是	R
fwVersion		string	--	--	--	--	256	是	R

命令列表

命令名称	Method
PUSH_COMMAND	

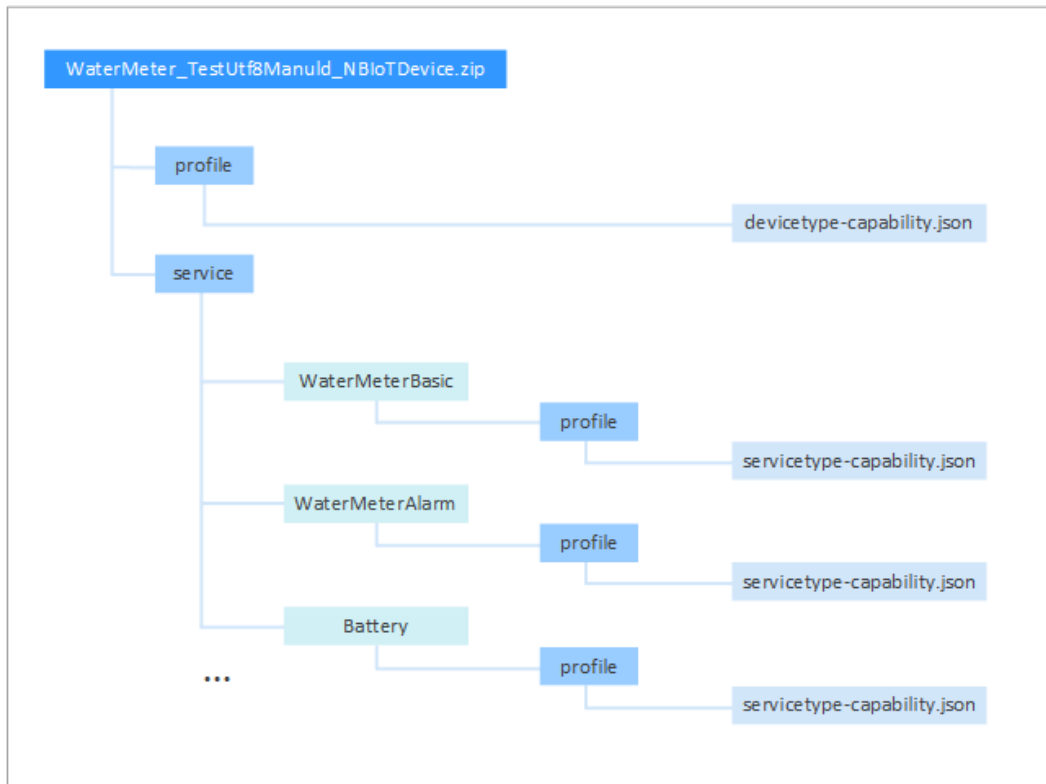
----结束

### 4.4.3 离线开发 Profile（联通用户专用）

非联通用户请查看[设备接入服务](#)。

Profile本质上就是一个deviceType-capability.json文件和若干个serviceType-capability.json文件，按照如下目录打包的一个zip包。其中WaterMeter是deviceType，TestUtf8Manuld是manufactureId，NBloTDevice是model，WaterMeterBasic/WaterMeterAlarm/Battery是服务名。





所以离线开发profile就是按照Profile编写规则和JSON格式规范在devicetype-capability.json中定义设备能力，在servicetype-capability.json中定义服务能力。因此离线开发Profile需要熟悉JSON的格式。

由于离线开发Profile文件相对在线开发比较耗时，因此推荐[在线开发Profile（联通用户专用）](#)。

## 命名规范

在Profile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和型号唯一标识一款产品，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。

## Profile 模板

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
厂商ID	manufacturerId	TestUtf8Manuld
厂商名称	manufacturerName	HZYB
型号	model	NBLoTDevice
协议类型	protocolType	CoAP

#### 设备的服务列表

服务描述	服务标识 (serviceld)	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

## 设备能力定义样例

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

```
{
  "devices": [
    {
      "manufacturerId": "TestUtf8Manuld",
      "manufacturerName": "HZYB",
      "model": "NBLoTDevice",
      "protocolType": "CoAP",
      "deviceType": "WaterMeter",
      "omCapability": {
        "upgradeCapability": {
          "supportUpgrade": true,
          "upgradeProtocolType": "PCP"
        }
      },
      "fwUpgradeCapability": {
```



字段	子字段		可选/必选	描述
	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。
	serviceTypeCapabilities		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。
	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。

字段	子字段	可选/必选	描述
	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。

## 服务能力定义样例

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
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```



字段	子字段				必选/可选	描述
	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。
		params			必选	命令包含的参数。
			paramName		必选	命令中参数的名字。
			dataType		必选	指示命令参数的数据类型。 取值范围：string、int、string list、decimal、DateTime、jsonObject、enum、boolean。 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	指示本命令是否必选，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。
			max		必选	指示最大值。 仅当dataType为int、decimal时生效。
			step		必选	指示步长。 暂不使用，填0即可。

字段	子字段			必选/可选	描述
			maxLength	必选	指示字符串长度。 仅当dataType为string、string list、DateTime时生效。
			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”
			enumList	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。
		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”。
			paras	必选	命令响应的参数。
			parameterName	必选	命令中参数的名字。
			dataType	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>



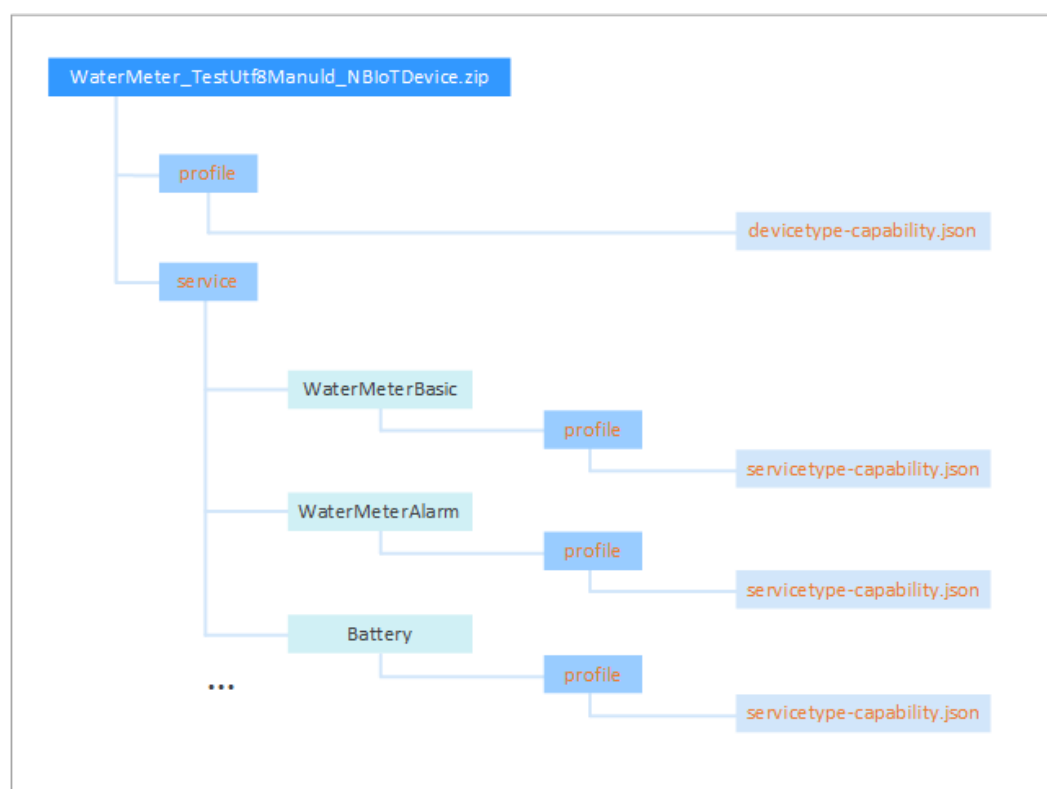
字段	子字段				必选/可选	描述
				required	必选	指示本命令响应是否必选，取值为true或false，默认取值false（非必选）。 目前本字段是非功能性字段，仅起到描述作用。
				min	必选	指示最小值。 仅当dataType为int、decimal时生效，逻辑大于等于。
				max	必选	指示最大值。 仅当dataType为int、decimal时生效，逻辑小于等于。
				step	必选	指示步长。 暂不使用，填0即可。
				maxLength	必选	指示字符串长度。 仅当dataType为string、string list、DateTime时生效。
				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”
				enumList	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。
	properties				必选	上报数据描述，每一个子节点为一条属性。
		propertyName			必选	指示属性名称。

字段	子字段			必选/可选	描述
		data Type		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。
		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。

字段	子字段			必选/可选	描述
		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。

## Profile 打包

Profile写作完成后，需要按如下层级结构打包：



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上图所示，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。
- 图中橙色字体的命名不能改动。
- 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": "NBloTDevice",
      "protocolType": "CoAP",
      "deviceType": "WaterMeter",
      "serviceTypeCapabilities": ****
    }
  ]
}
```

- 图中的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。

#### 4.4.4 导出和导入 Profile

联通用户专用，非联通用户请查看[设备接入服务](#)。

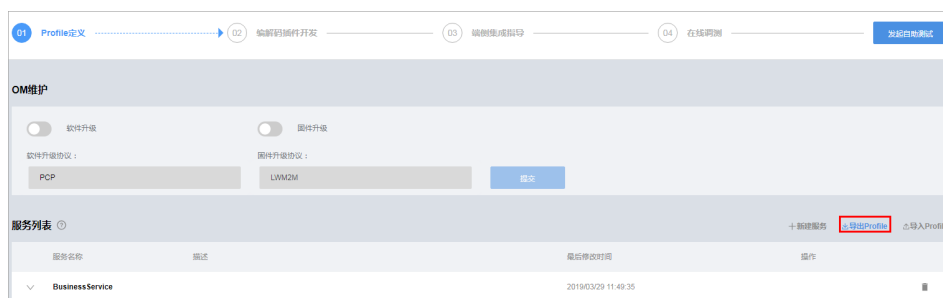
Profile作为一种资源，可以从物联网平台导出，也可以导入到物联网平台。

- 当产品开发完成并测试验证后，需要将在线开发的Profile移植时，则可以将Profile导出到本地。
- 当您已经有完备的Profile时（线下开发或从其他项目/平台导出），可以将Profile直接导入到开发中心或“设备管理服务控制台”。

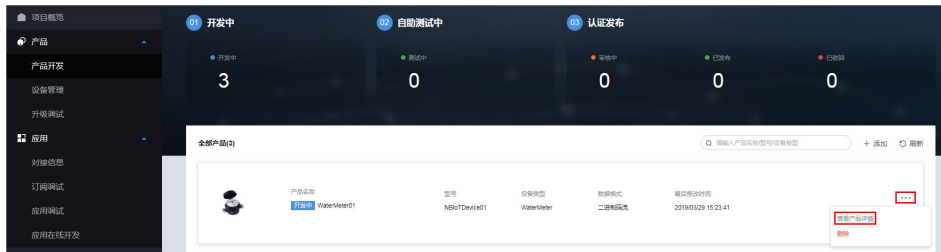
#### 导出 Profile

当产品开发完成并测试验证后，需要将在线开发的Profile移植时，则可以将Profile导出到本地。

- 在开发中心的“Profile定义”界面导出
  - a. 在开发中心的具体项目中，选择“产品 > 产品开发”，单击进入某个产品，在“Profile定义”界面，点击“导出Profile”。



- b. 选择Profile的存放路径，将Profile下载到本地。
- 在开发中心的“产品详情”界面导出
    - a. 在开发中心的具体项目中，选择“产品 > 产品开发”，点击产品右侧“...”按钮，然后点击“查看产品详情”，进入“产品详情”界面。



- b. 在“产品详情”界面，在右侧相关资料区域，点击“下载”，将Profile下载到本地。

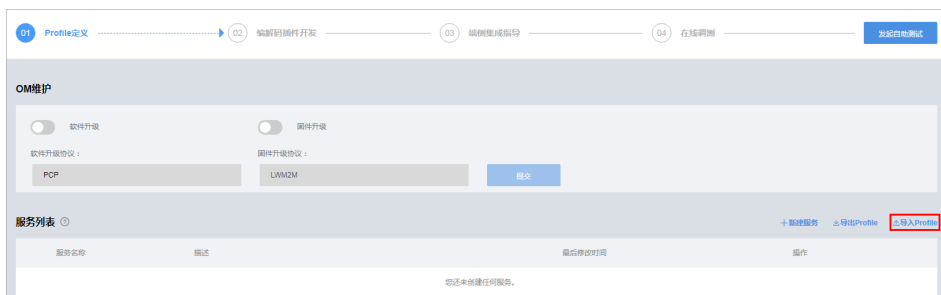


## 导入 Profile

当您已经有完备的Profile时（线下开发或从其他项目/平台导出），可以将Profile直接导入到开发中心或“设备管理服务控制台”。


- 在开发中心导入

- a. 在开发中心的具体项目中，选择“产品 > 产品开发”，单击进入某个产品，在“Profile定义”界面，点击“导入Profile”。



- b. 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。

- 在“设备管理服务控制台”导入

- 登录“设备管理服务控制台”，单击左下角的  切换左侧菜单，打开“产品模型”页面，点击页面右上角的“新增产品模型 > 本地导入”。

产品名称	型号	设备类型	厂商名称	协议类型	创建时间	详情	删除
StreetLight_hw_SL01	SL01	StreetLight	hw	CoAP	2019-05-06 15:18:15		

- 在弹窗中输入产品名称并选择要上传的Profile文件，单击“确定”，导入Profile文件。



## 4.5 开发编解码插件

### 4.5.1 什么是编解码插件（联通用户专用）

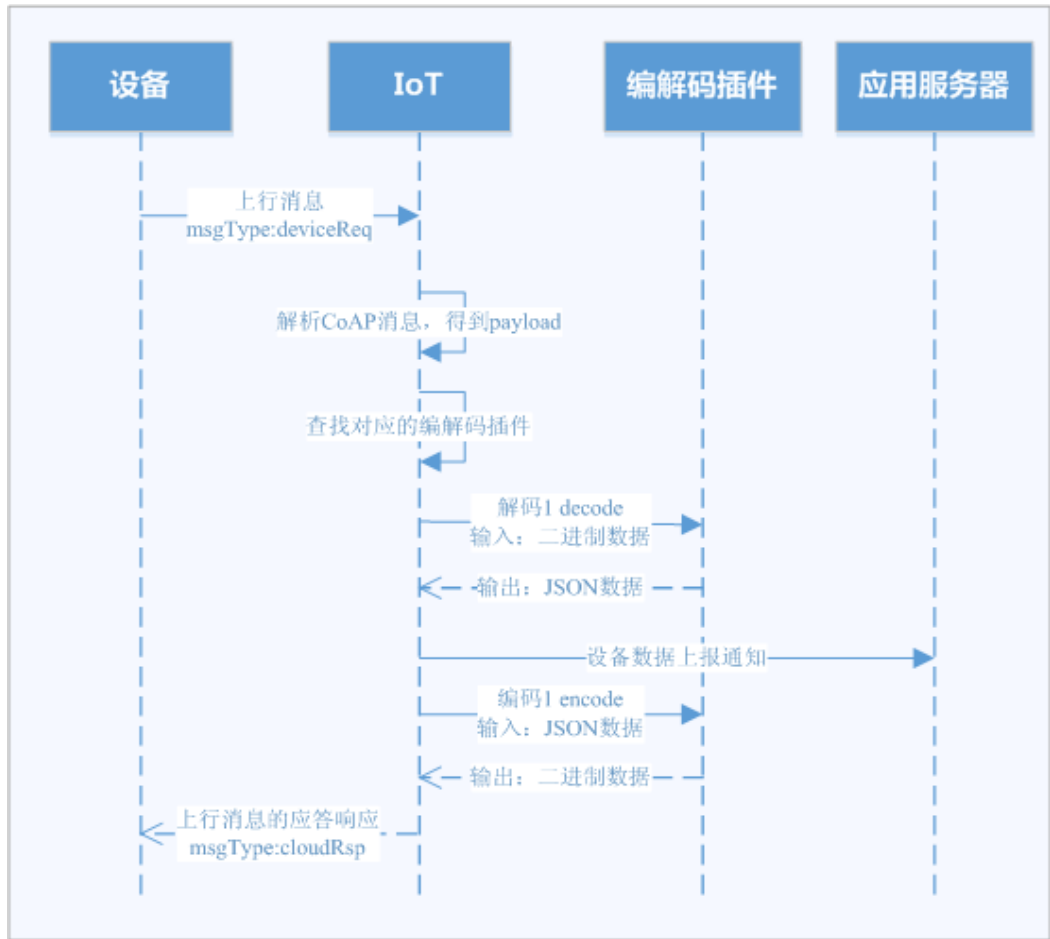
非联通用户请查看[设备接入服务](#)。

一款产品的设备上报数据时，如果“数据格式”为“二进制码流”，则该产品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，您需要开发编码插件，供物联网平台调用，以完成二进制格式和JSON格式的转换。



## 数据上报流程

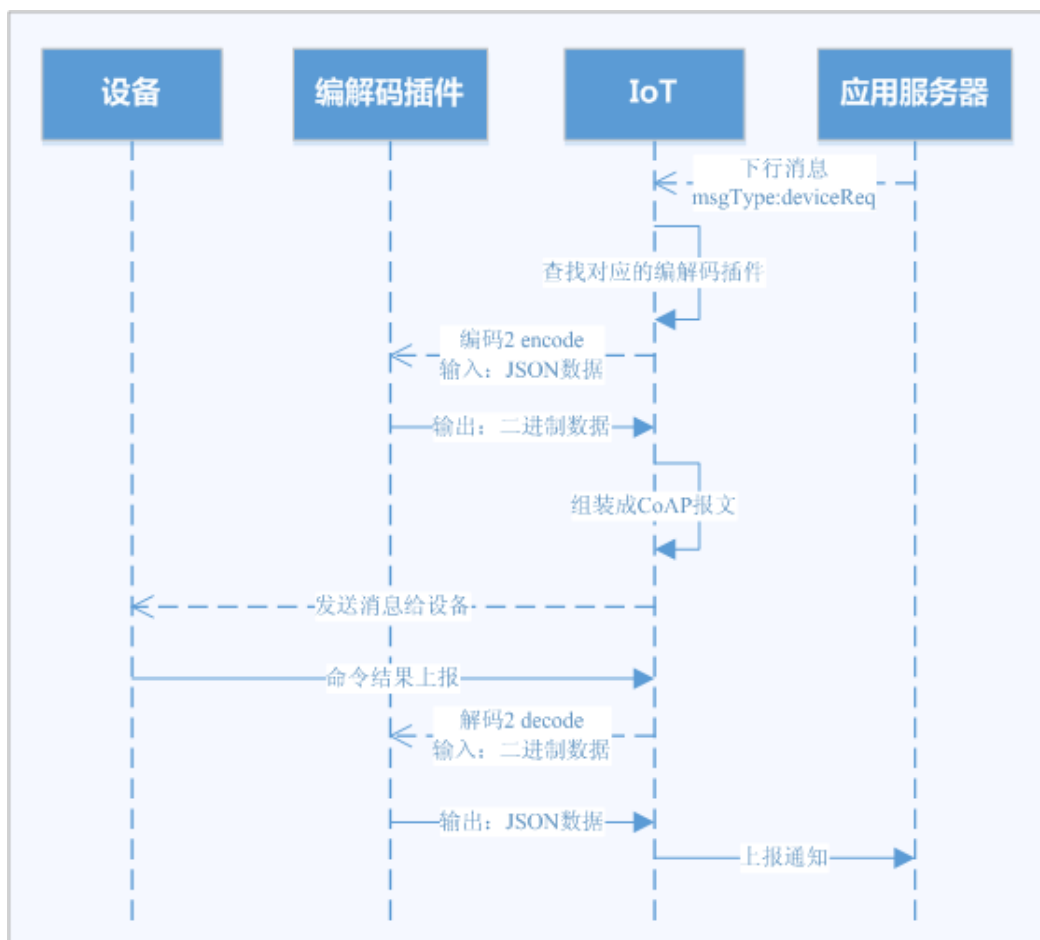


在数据上报流程中，有两处需要用到编解码插件：

- 将设备上报的二进制码流解码成JSON格式的数据，发送给应用服务器。
- 将应用服务器响应的JSON格式数据编码成二进制码流，下发给设备。



## 命令下发流程



在命令下发流程中，有两处需要用到编解码插件：

- 将应用服务器下发的JSON格式数据编码成二进制码流，下发给设备。
- 将设备响应的二进制码流解码成JSON格式的数据，上报给应用服务器。

## 在线开发和离线开发

编解码插件的开发手段有在线开发和离线开发两种，由于插件离线开发较为复杂，且耗时比较长，我们推荐使用开发中心**在线开发编解码插件**。

- **在线开发**是指借助开发中心，通过可视化的方式快速开发一款产品的编解码插件。
- **离线开发**是指使用编解码插件的Java代码Demo进行二次开发，实现编解码功能、完成插件打包和质检等。

### 4.5.2 在线开发插件（联通用户专用）

非联通用户请查看**设备接入服务**。

借助开发中心，我们可以通过可视化的方式快速开发一款产品的编解码插件。在自定义新建产品时，如果选择使用系统模板，部分模板会包含编解码插件，您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，则需要完成编解码插件的开发。

编解码插件的开发需要基于Profile定义的设备能力进行开发，本节首先以一个烟感设备的例子讲解如何开发一个支持数据上报和命令下发的编解码插件，然后再以四个场景举例说明如何完成复杂的插件开发以及调试。

- [数据上报和命令下发](#)
- [多条数据上报消息](#)
- [字符串及可变长字符串的编解码插件在线开发](#)
- [数组及可变长数组数据类型](#)
- [含命令执行结果的编解码插件](#)

## 数据上报和命令下发

### 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。

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> + 添加属性			
level	数据类型: int, 范围: 0 ~ 3, 步长: --, 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
temperature	数据类型: int, 范围: 0 ~ 1000, 步长: --, 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
<b>命令列表</b> + 添加命令			
SET_ALARM			
<b>下发命令字段</b> + 添加下发字段			
value	数据类型: int, 范围: 0 ~ 3, 步长: --, 单位: --	是否必填: <input checked="" type="checkbox"/>	
<b>响应命令字段</b> + 添加响应字段			
您还未创建任何响应命令。			

### 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。

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 ⓘ  
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 \* SET\_ALARM

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

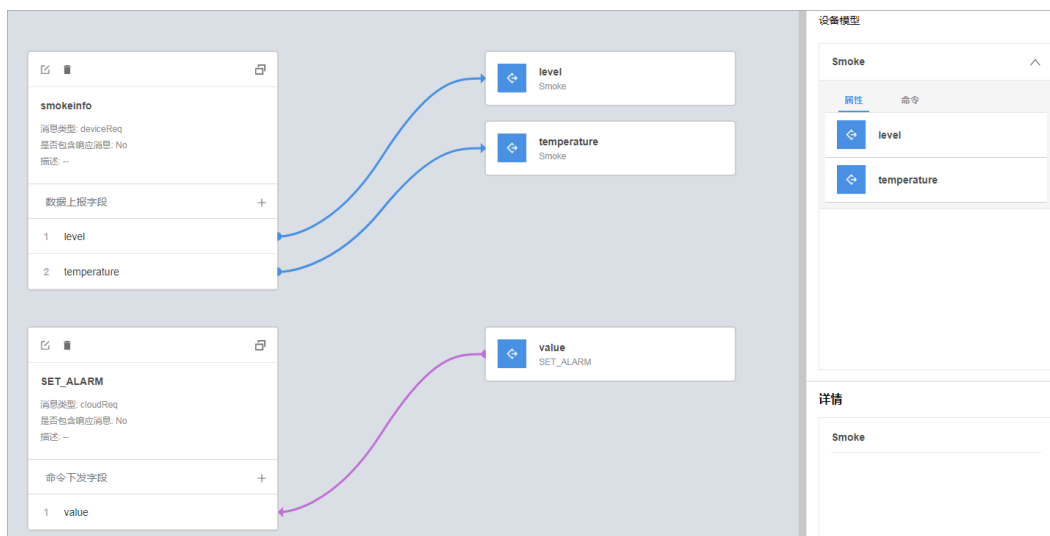
默认值 ?

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。

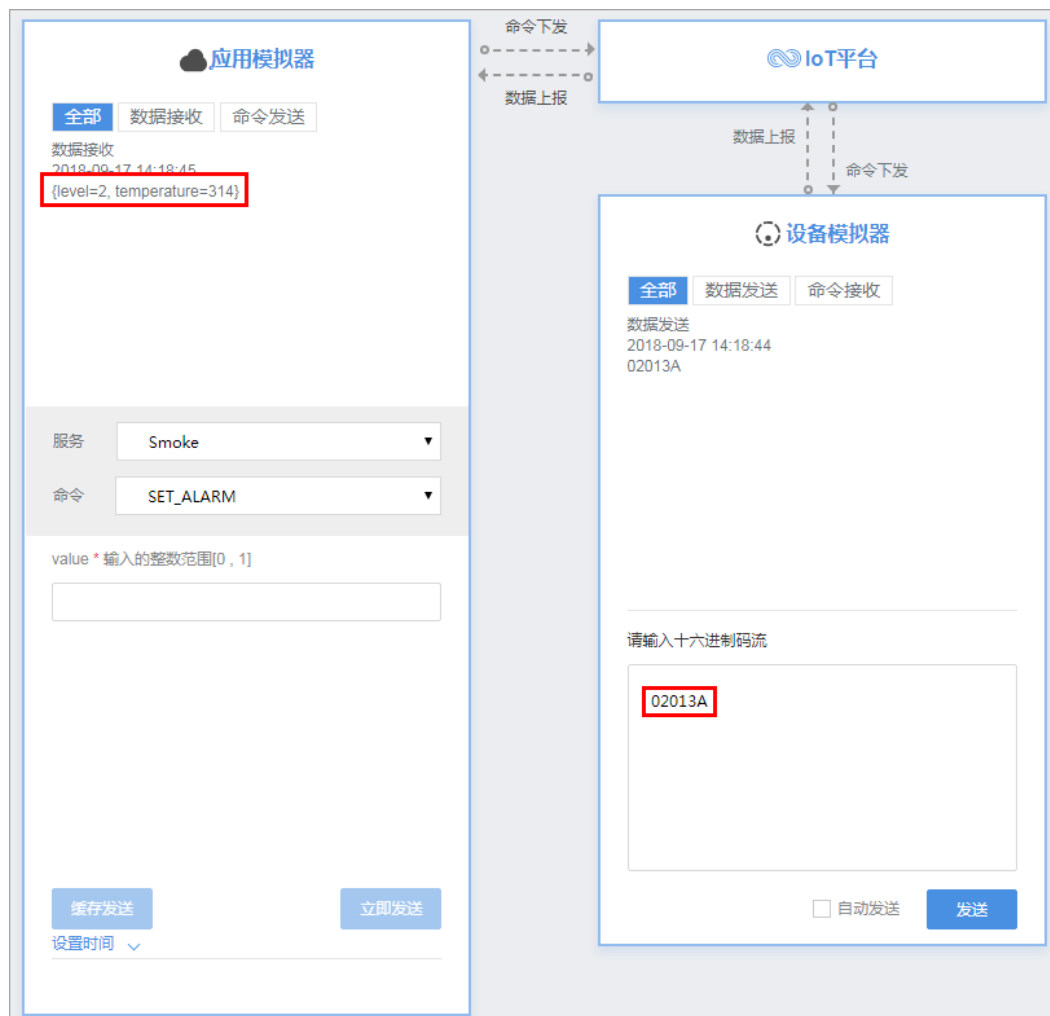


勾选“没有真实的物理设备”，点击“创建”。



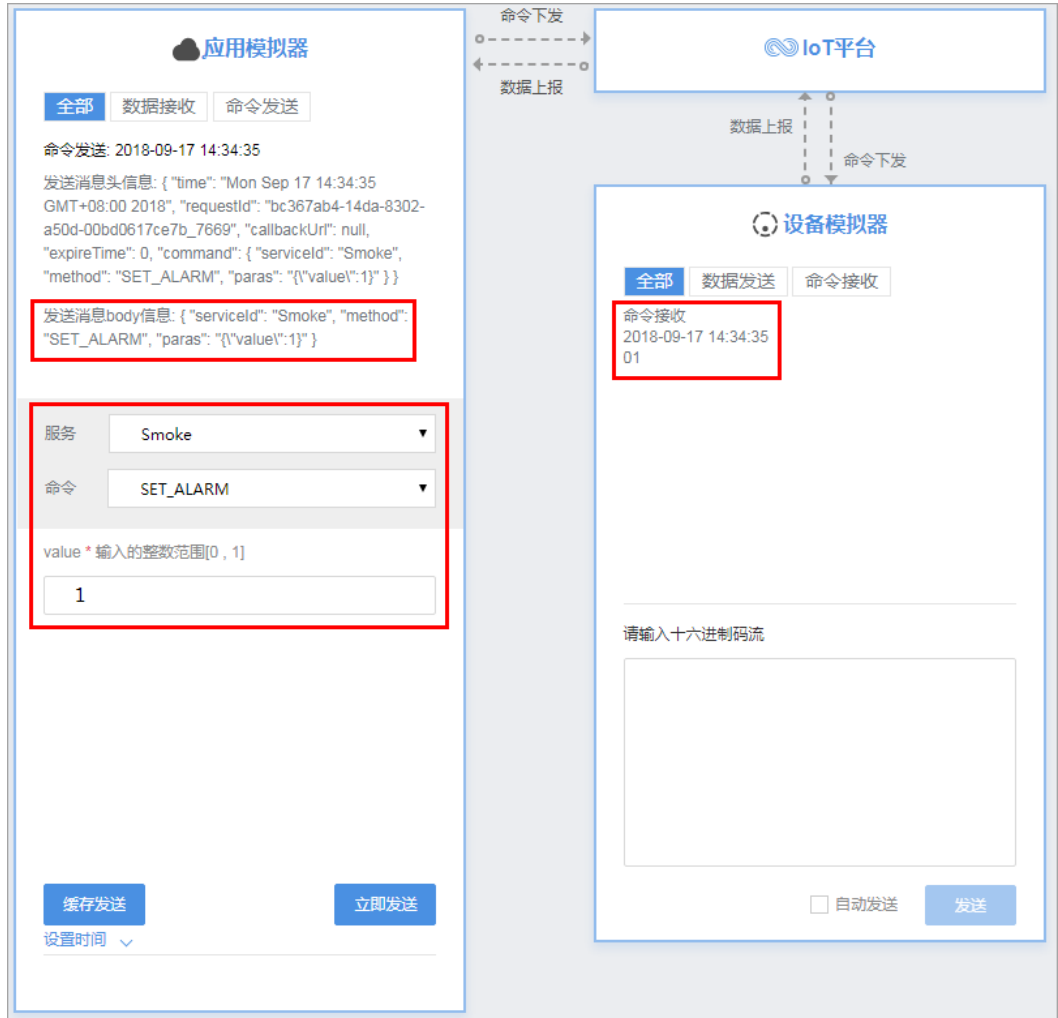
**步骤2** 使用设备模拟器进行数据上报。十六进制码流示例：02013A。02表示火灾级别，长度为1个字节；013A表示温度，长度为2个字节。

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：{ "serviceld": "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

## 多条数据上报消息

如果该烟感设备需要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，则按照以下步骤创建消息。

### 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。



新增消息

基本信息

消息名 \*  
smokeinfo

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，数据上报消息有两种，所以需要messageId来标识消息种类。
- “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 ×

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**
✕

标记为地址域 ?

**\* 名字**

描述

数据类型

**\* 长度** ?

默认值 ?

偏移值 ?

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。

The image shows a '新增消息' (Add Message) dialog box with the following fields and options:

- 消息名 \***: A text input field containing 'temperature'.
- 消息描述**: A large empty text area.
- \* 消息类型**: Radio buttons for '数据上报' (selected) and '命令下发'.
- 添加响应字段
- 字段**: A section with a '+ 添加字段' (Add Field) link.
- Buttons: '完成' (Done) and '取消' (Cancel).

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 ×

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 \*  
SET\_ALARM

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**

描述

数据类型

int8u(8位无符号整型) ▼

**\* 长度** ?

默认值 ?

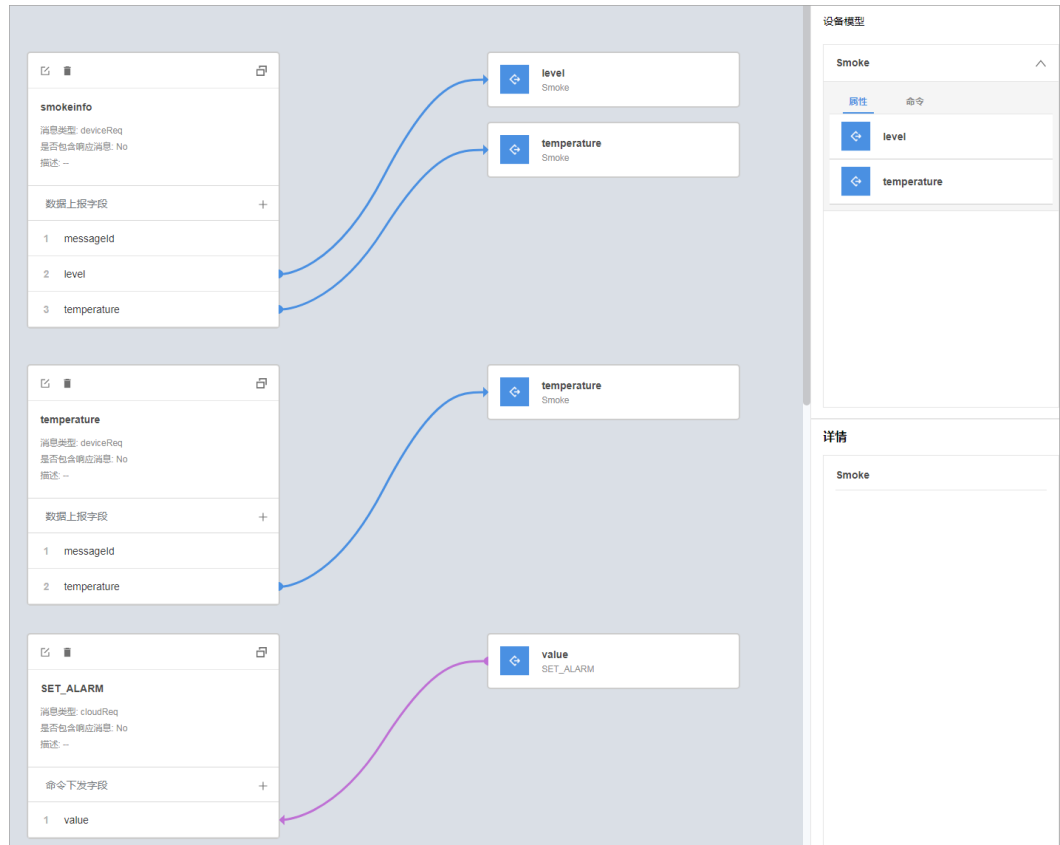
偏移值 ?

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。

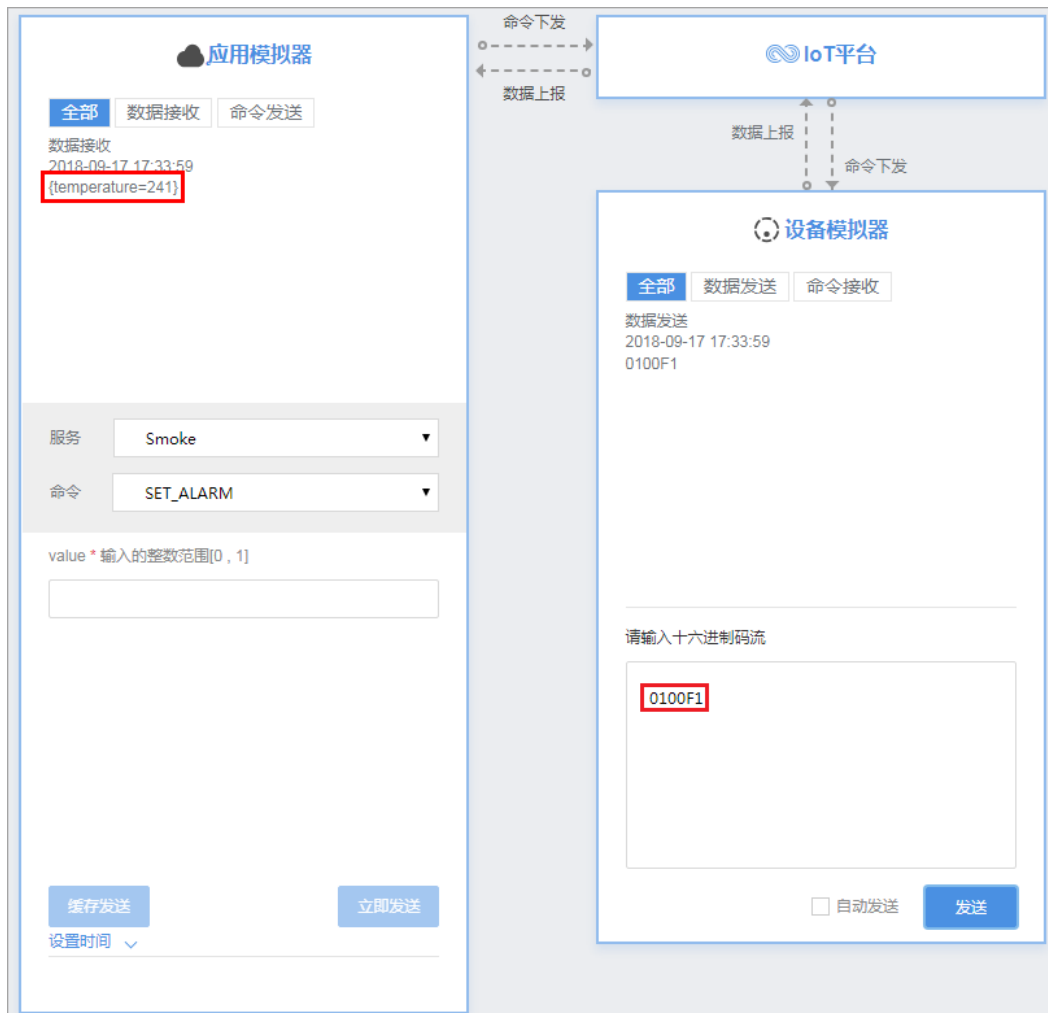
十六进制码流示例：000100F1。00表示messageId，此消息上报火灾等级和温度；01表示火灾级别，长度为1个字节；00F1表示温度，长度为2个字节。

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。



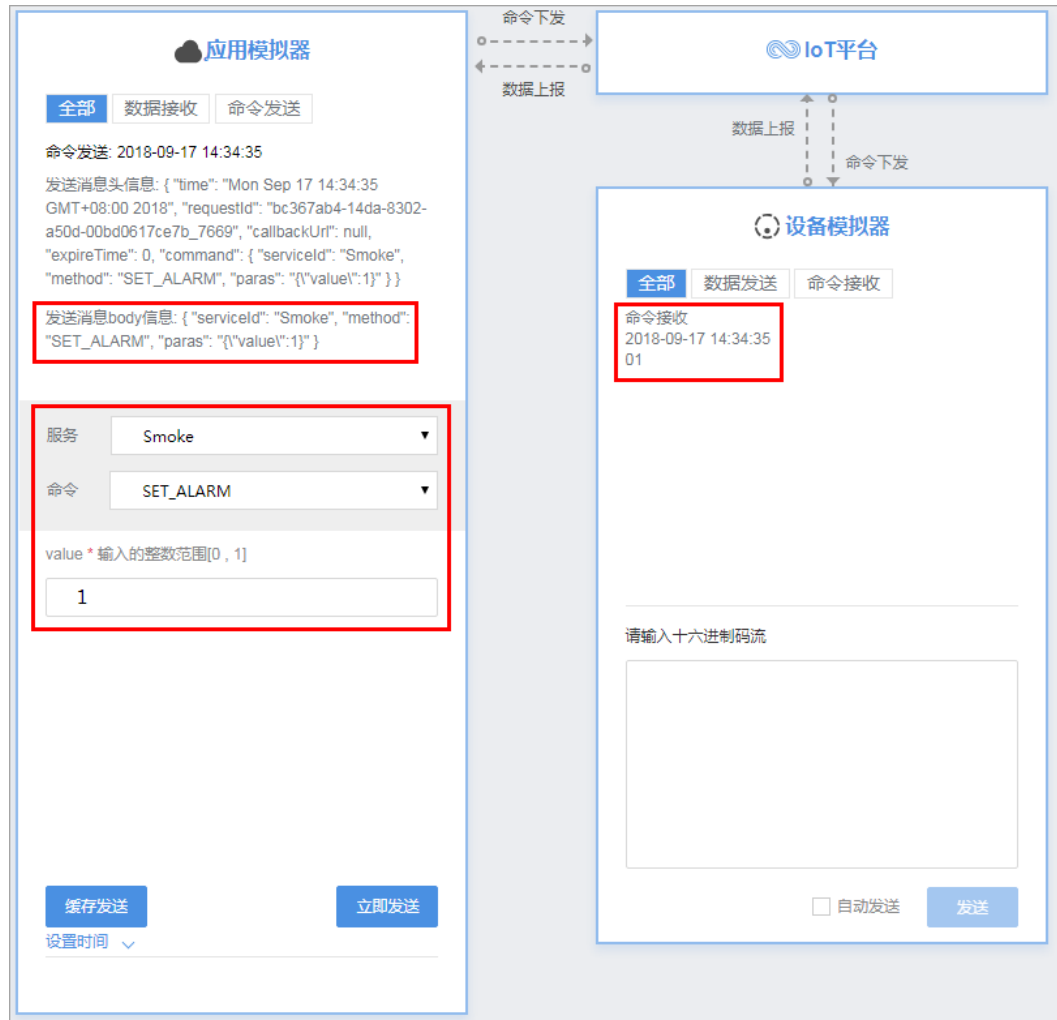
十六进制码流示例：0100F1。01表示messageId，此消息只上报火灾温度；00F1表示温度，长度为2个字节。

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：{ "serviceld": "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

## 字符串及可变量字符串的编解码插件在线开发

如果该烟感设备需要支持描述信息上报功能，描述信息支持字符串和可变量字符串两种类型，则按照以下步骤创建消息。

### 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	
属性列表 + 添加属性			
level	数据类型: int, 范围: 0 ~ 3, 步长: --, 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
temperature	数据类型: int, 范围: 0 ~ 1000, 步长: --, 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
other_info	数据类型: string, 长度: 100, 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R

### 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，上报字符串类型的描述信息。

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 ✕

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

other\_info

描述

数据类型

string(字符串类型) ▼

\* 长度 ?

6

默认值 ?

偏移值 ?

1-7

完成 取消

**步骤3** 配置数据上报消息，上报可变长度字符串类型的描述信息。



The screenshot shows a 'New Message' (新增消息) dialog box with the following elements:

- Header:** 新增消息 (Close button)
- Section: 基本信息**
  - 消息名 (Message Name):** Input field containing 'other\_info\_2'.
  - 消息描述 (Message Description):** Text area.
  - 消息类型 (Message Type):** Radio buttons for '数据上报' (selected) and '命令下发'.
  - 添加响应字段 (Add Response Field):** Unchecked checkbox.
- Section: 字段 (Fields)**
  - + 添加字段 (Add Field):** Link to add new fields.
- Buttons:** 完成 (Complete) and 取消 (Cancel).

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**
✕

标记为地址域 ?

**\* 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

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 ×

标记为地址域 ?

\* 名字

other\_info

描述

数据类型

varstring(可变长度字符串类型) ▼

\* 长度关联字段 ?      \* 长度关联字段差值 ?

length ▼      0

数值长度 ?      \* 默认值 ?

1     

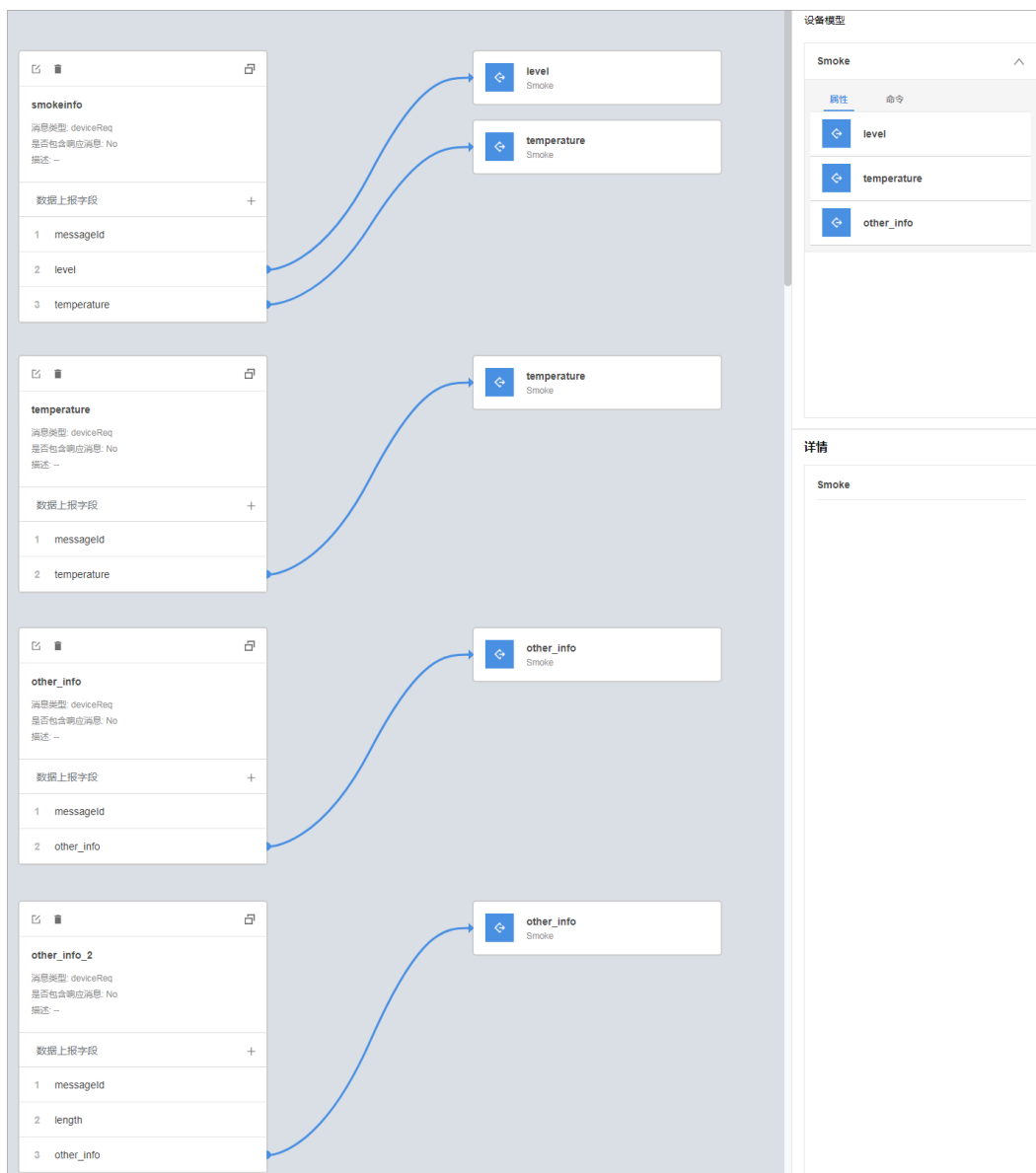
掩码 ?

偏移值 ?

2-3

完成      取消

**步骤4** 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。



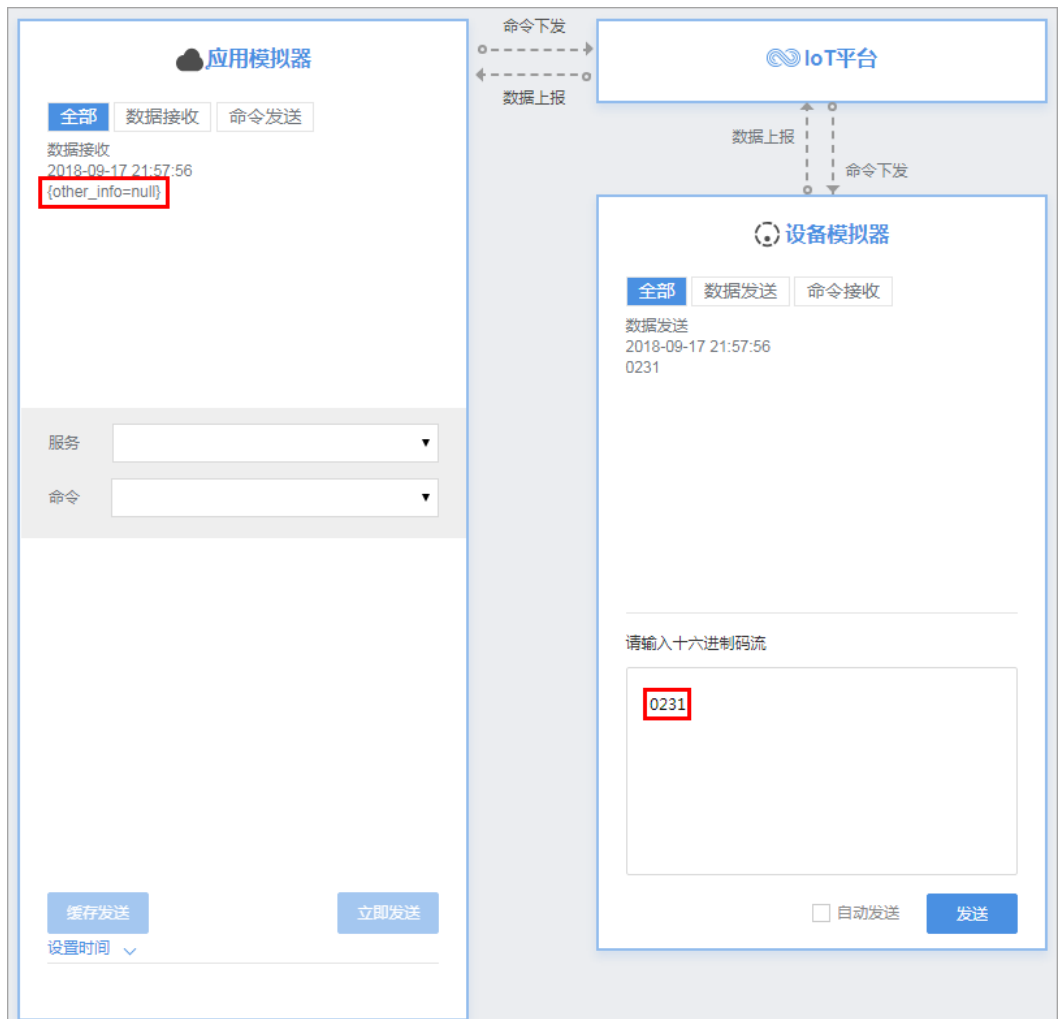
勾选“没有真实的物理设备”，点击“创建”。



**步骤2** 使用设备模拟器上报字符串类型的描述信息。

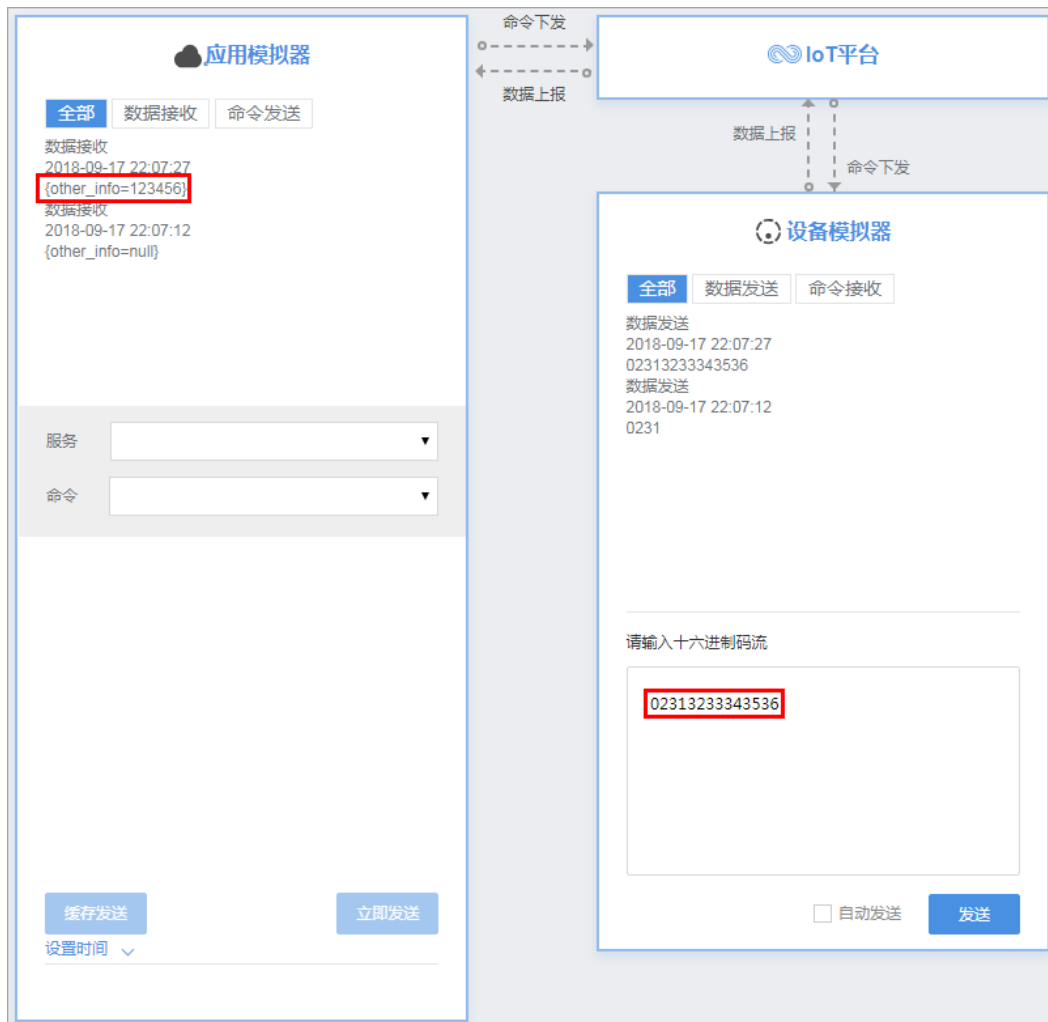
十六进制码流示例：0231。02表示messageId，此消息上报字符串类型的描述信息；31表示描述信息，长度为1个字节。

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。



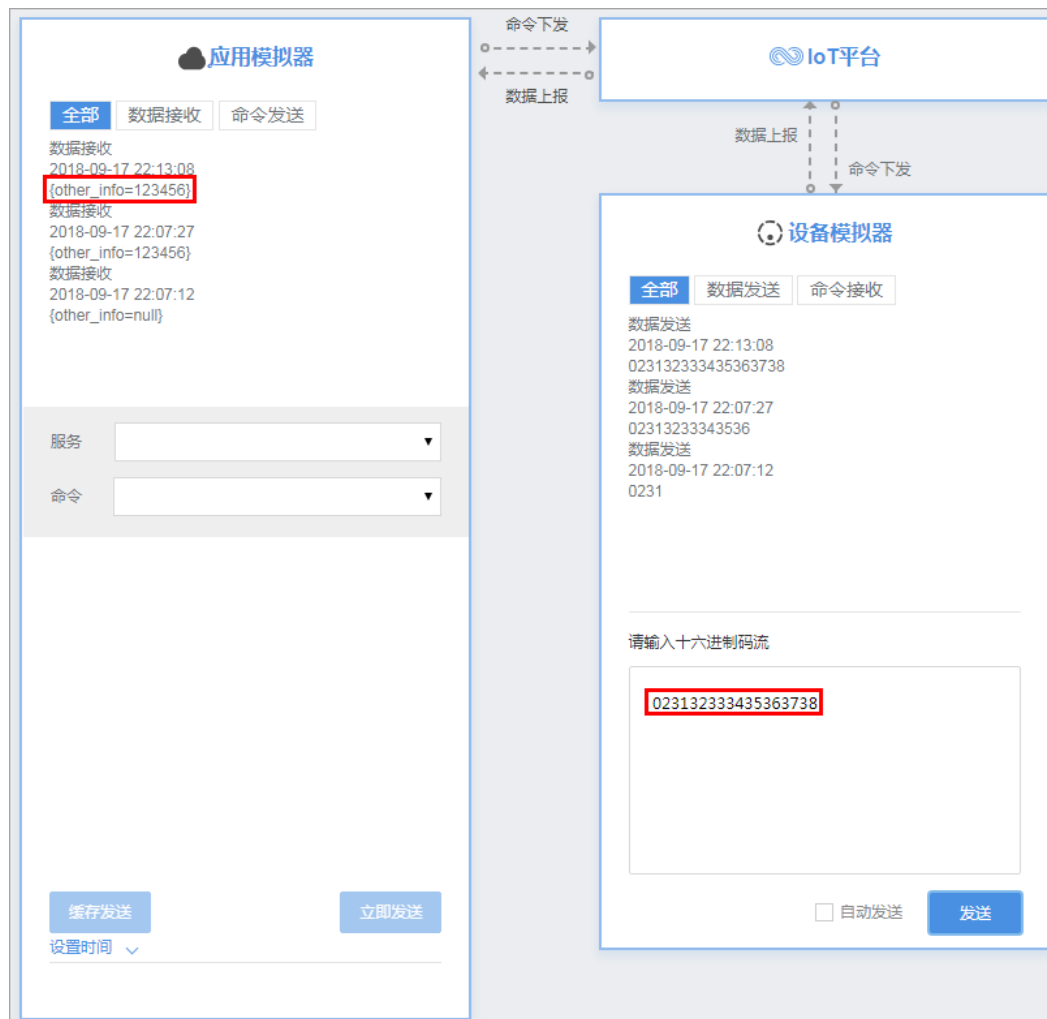
十六进制码流示例：02313233343536。02表示messageId，此消息上报字符串类型的描述信息；313233343536表示描述信息，长度为6个字节。

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。

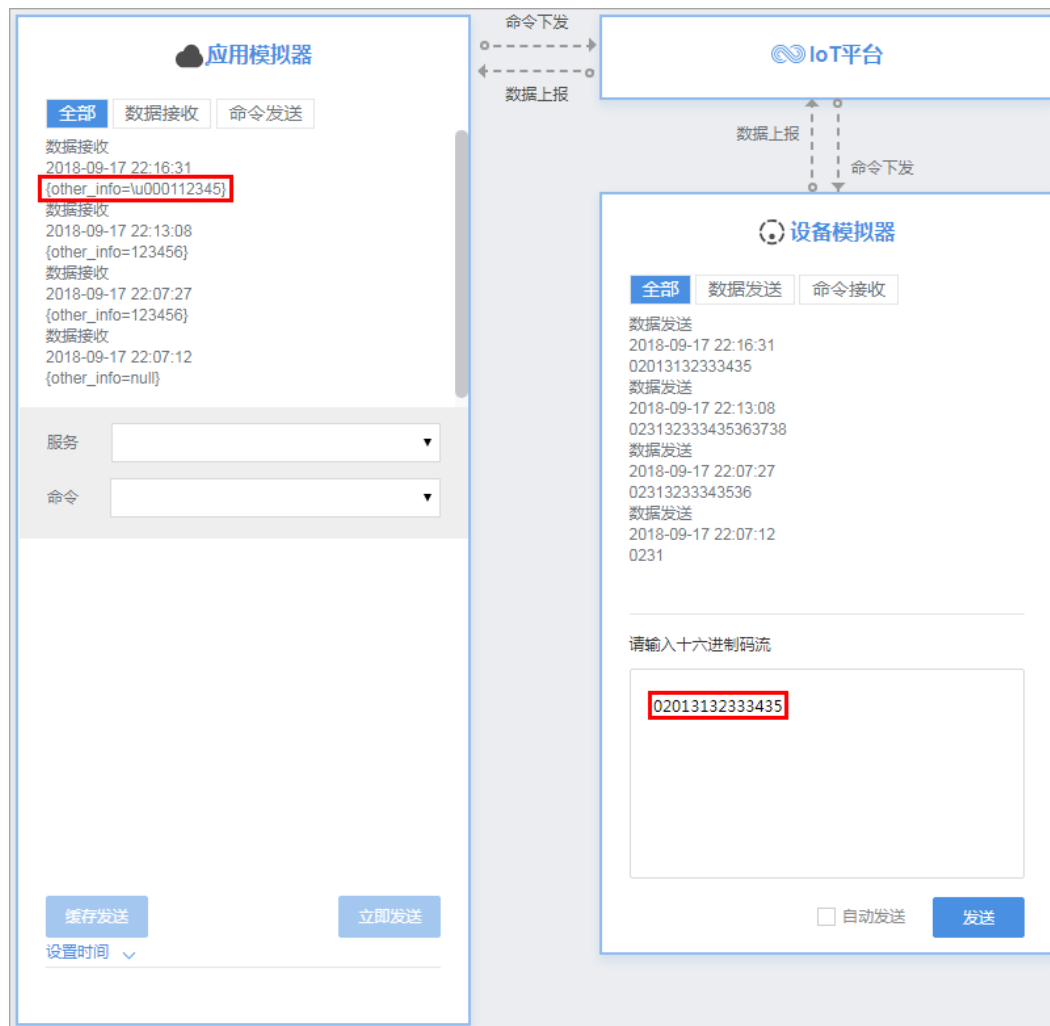
在“应用模拟器”区域查看数据上报的结果：{other\_info=123456}。描述信息长度超过6个字节，编解码插件截取前6个字节进行解析。



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。

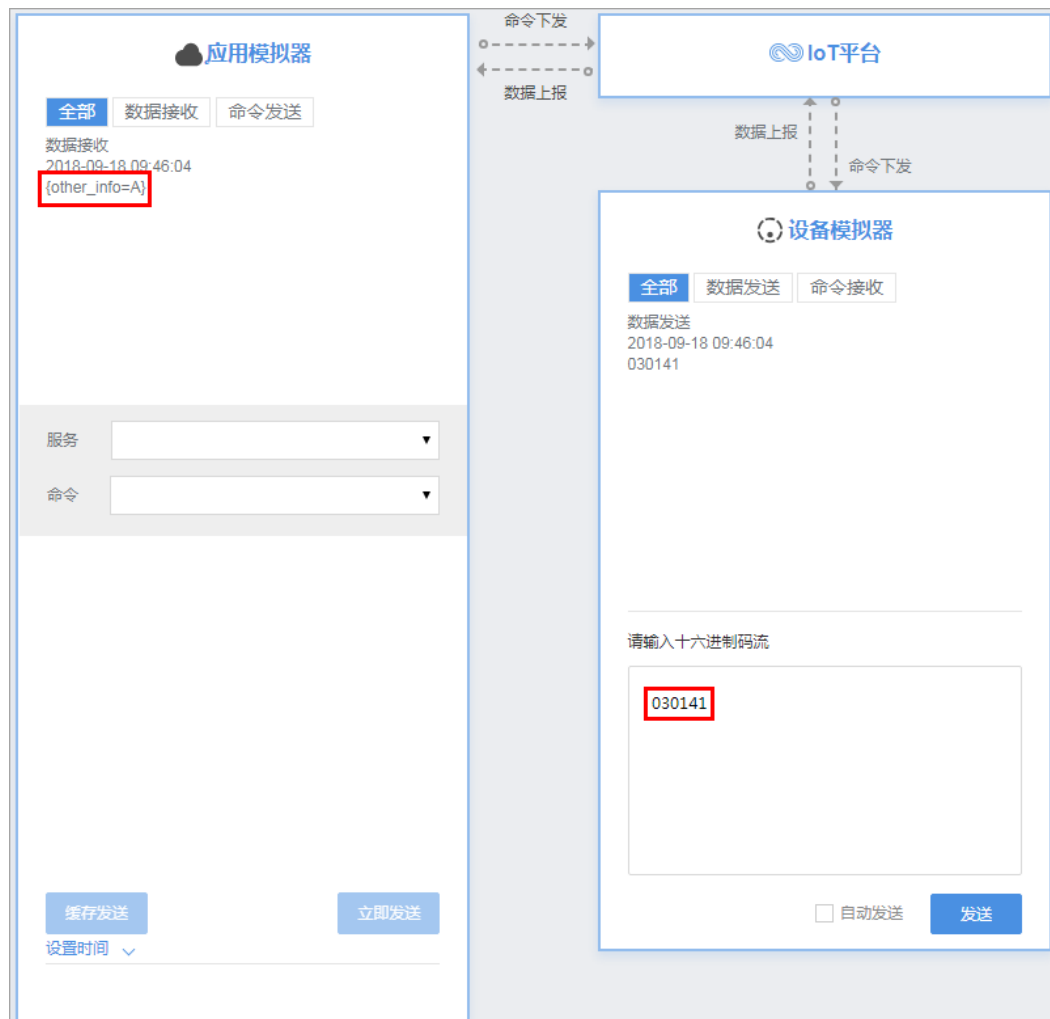




**步骤3** 使用设备模拟器上报可变长度字符串类型的描述信息。

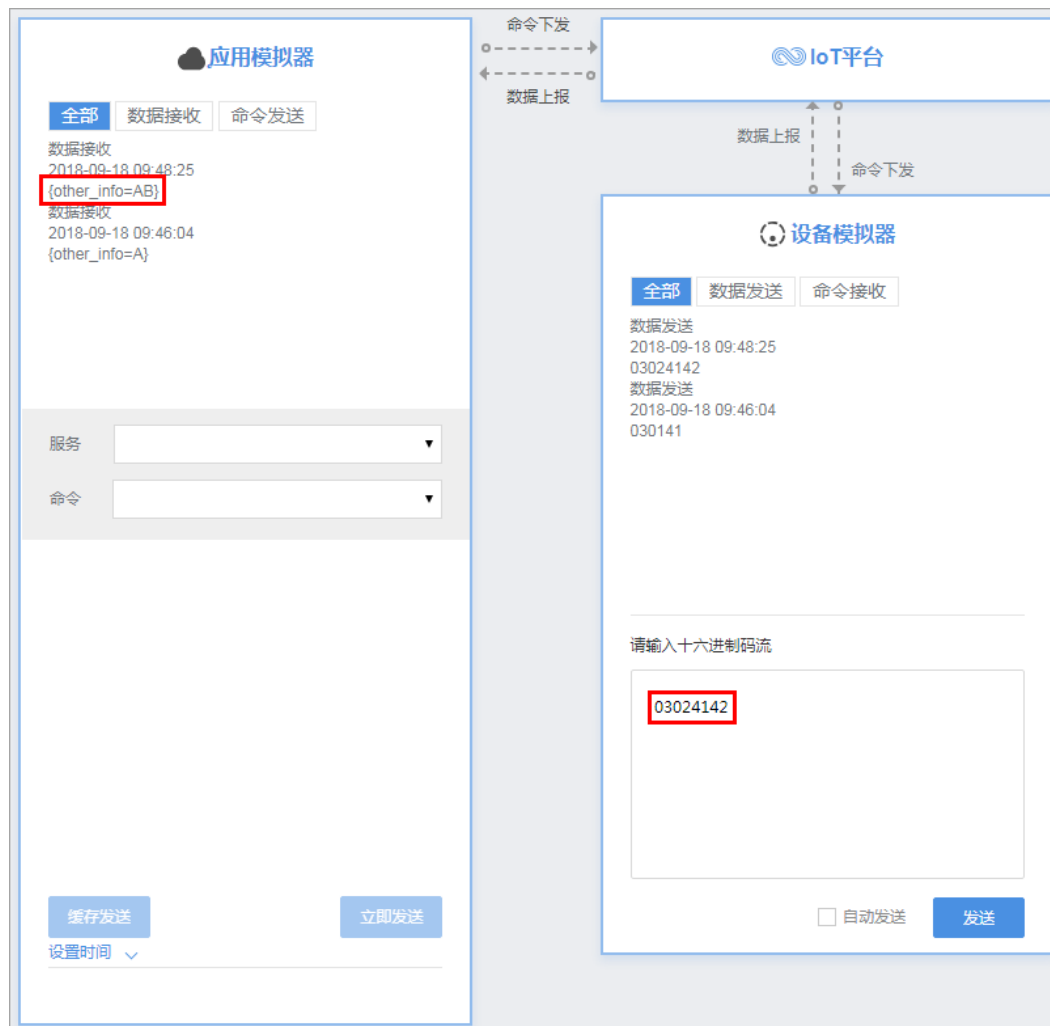
十六进制码流示例：030141。03表示messageId，此消息上报可变长度字符串类型的描述信息；01表示描述信息长度（1个字节），长度为1个字节；41表示描述信息，长度为1个字节。

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。



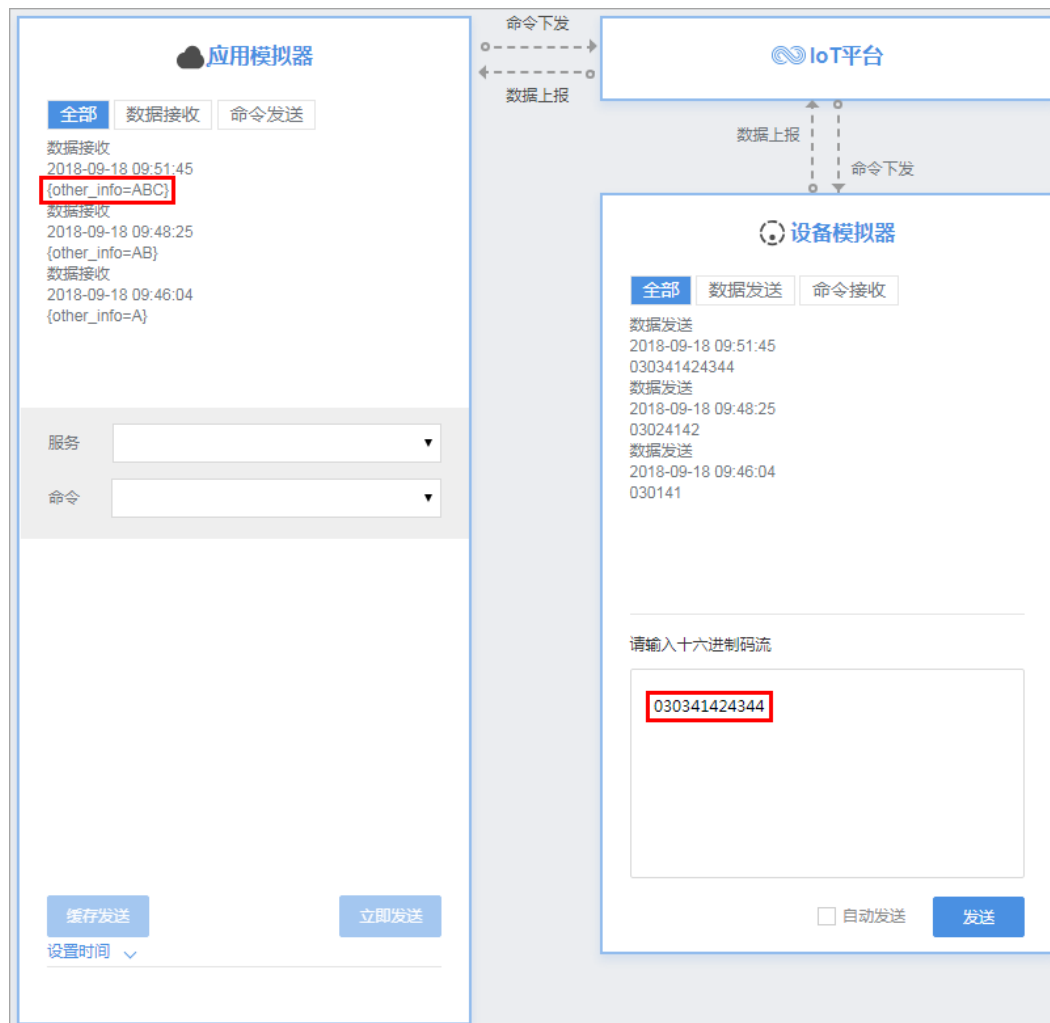
十六进制码流示例：03024142。03表示messageId，此消息上报可变长度字符串类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；4142表示描述信息，长度为2个字节。

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。



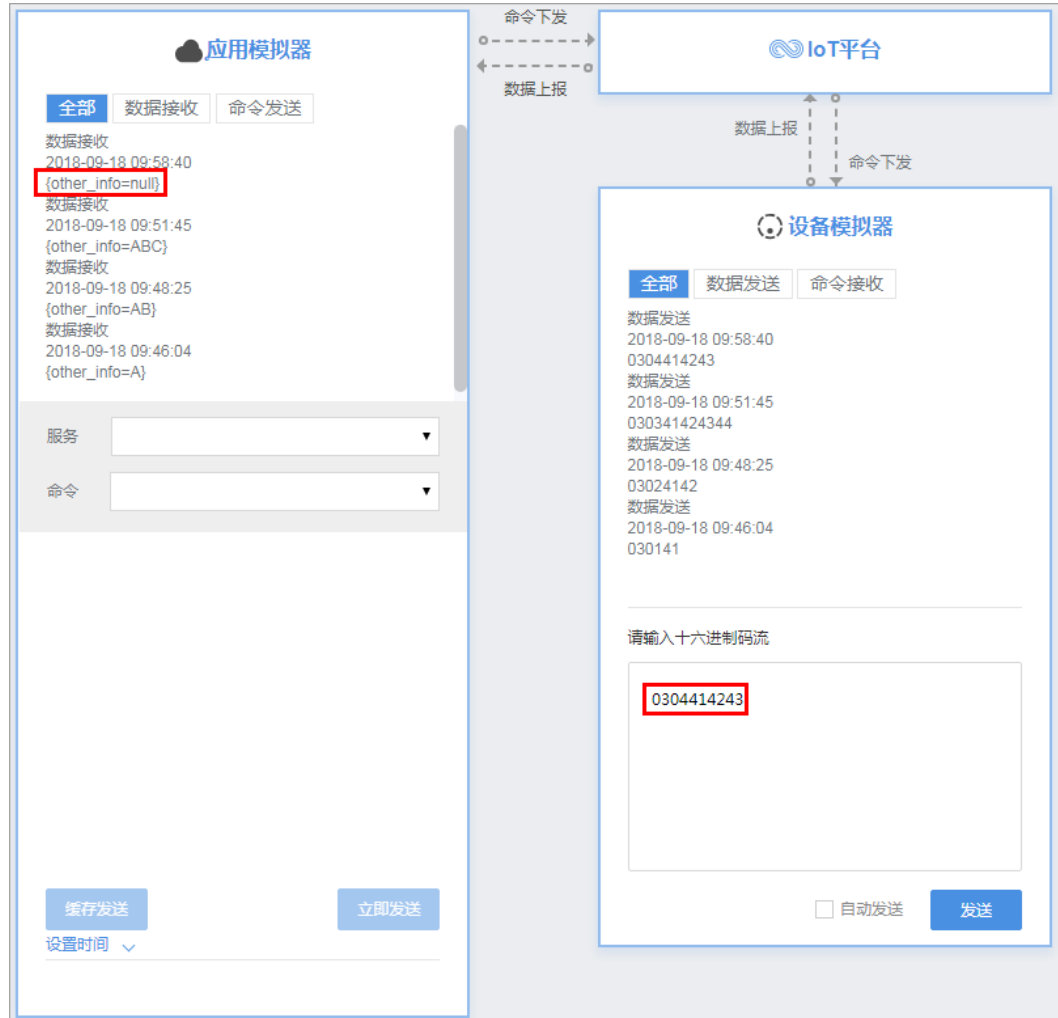
十六进制码流示例：030341424344。03表示messageId，此消息上报可变长度字符串类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；41424344表示描述信息，长度为4个字节。

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。

### 数组及可变长数组数据类型

如果该烟感设备需要支持描述信息上报功能，描述信息描述信息支持数组和可变长度数组两种类型，则按照以下步骤创建消息。

## 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		2019/09/17 10:25:15	
<b>属性列表</b> <span style="float: right;">+ 添加属性</span>			
level	数据类型: int 范围: 0 ~ 3 步长: -- 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
temperature	数据类型: int 范围: 0 ~ 1000 步长: -- 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R
other_info	数据类型: string 长度: 100 单位: --	是否必填: <input checked="" type="checkbox"/>	访问模式: R

## 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](#)

---

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 ✕

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3** 配置数据上报消息，上报可变长度数组类型的描述信息。



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

int8u(8位无符号整型) ▼

**\* 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     

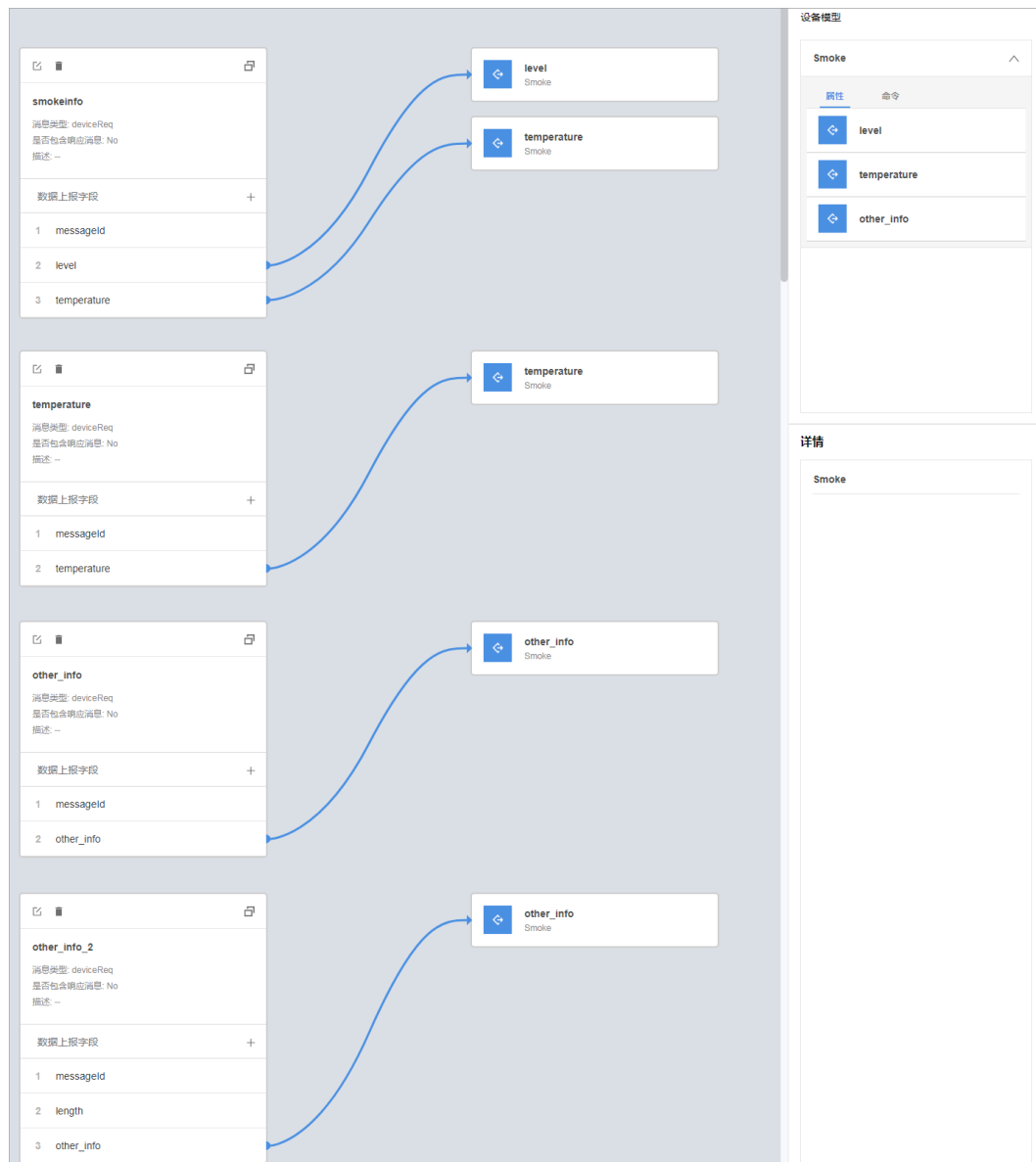
掩码 ?

偏移值 ?

2-3

完成      取消

**步骤4** 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。



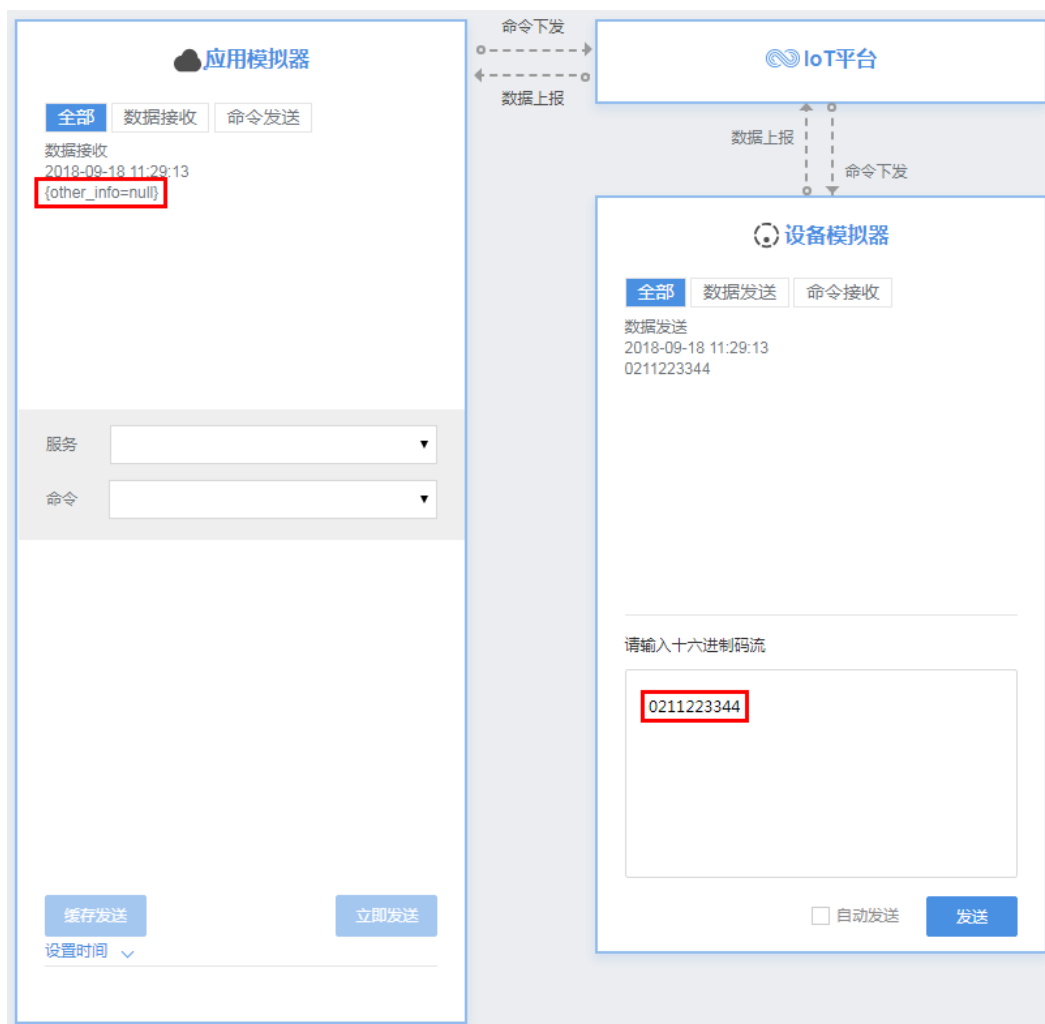
勾选“没有真实的物理设备”，点击“创建”。



**步骤2** 使用设备模拟器上报数组类型的描述信息。

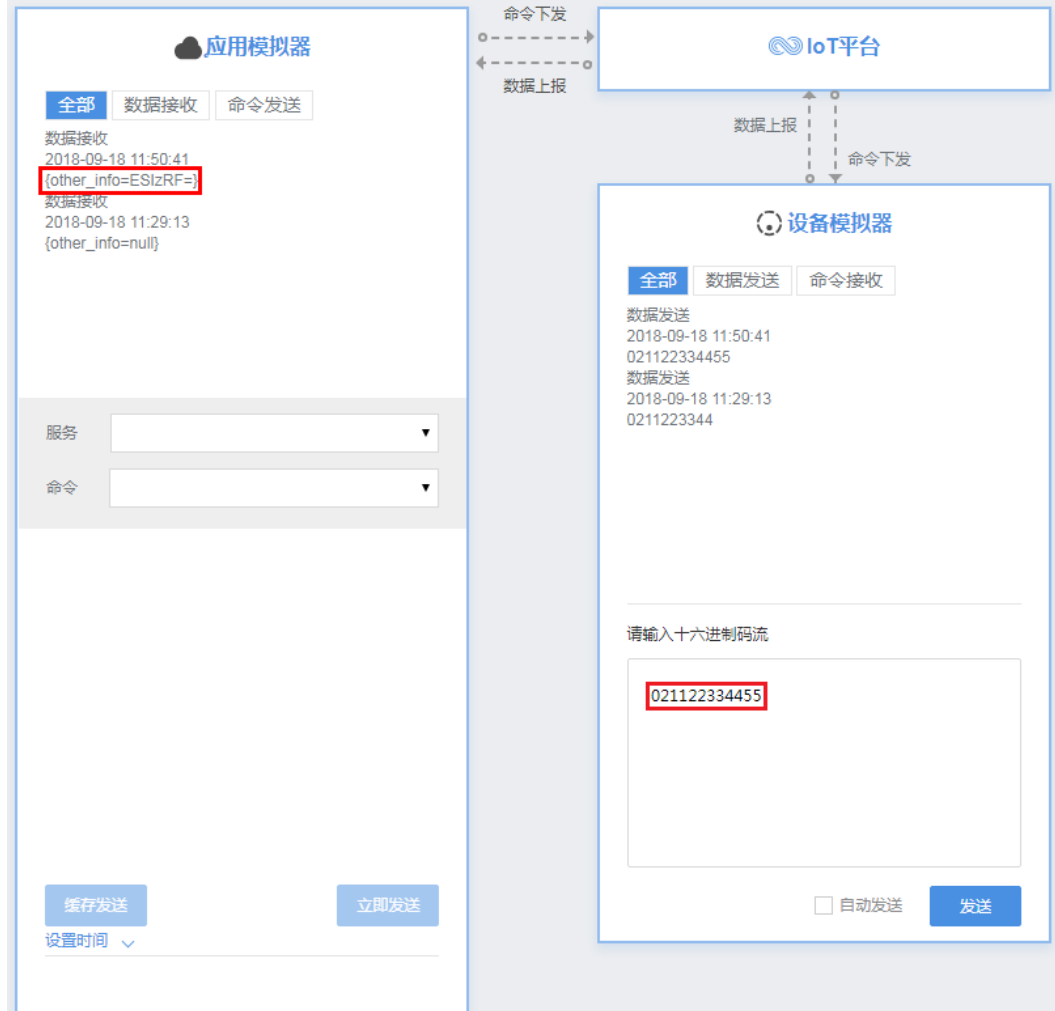
十六进制码流示例：0211223344。02表示messageId，此消息上报数组类型的描述信息；11223344表示描述信息，长度为4个字节。

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。



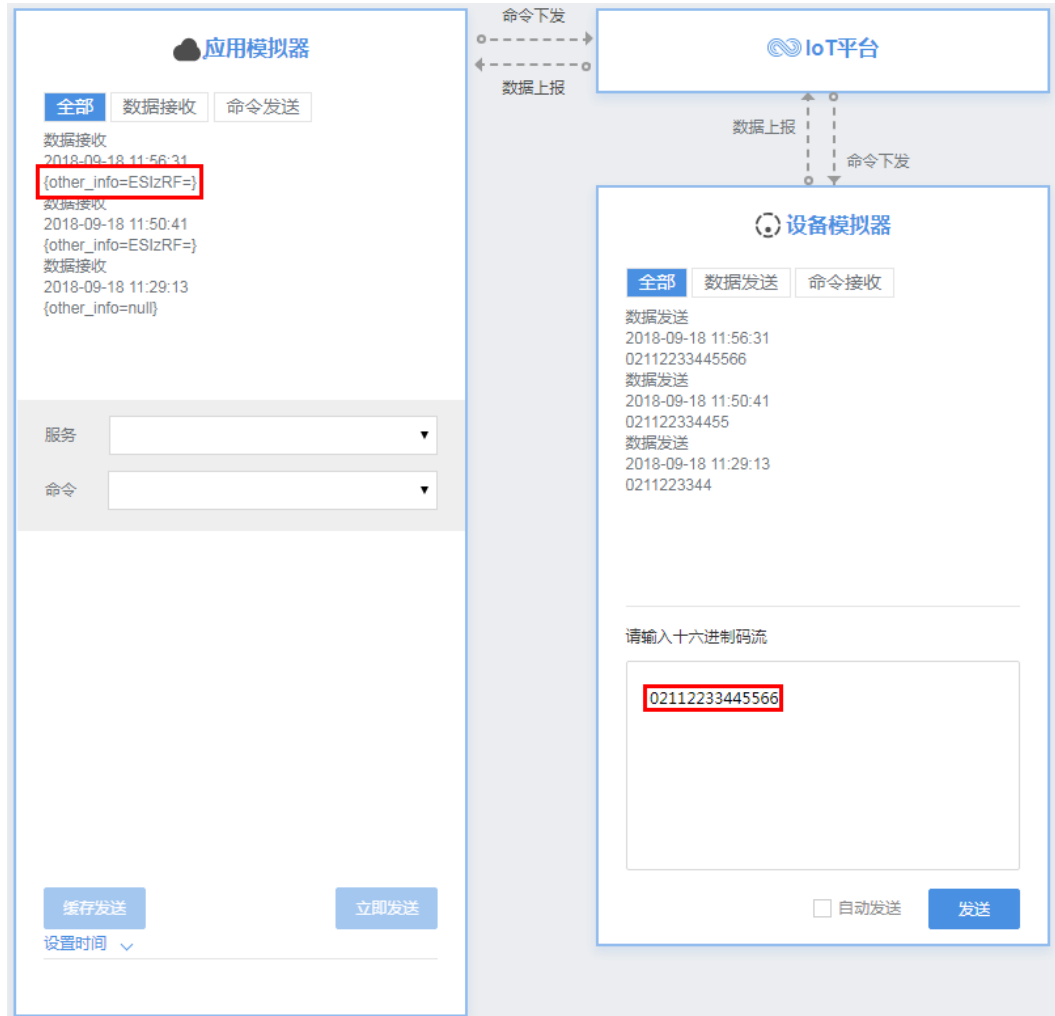
十六进制码流示例：021122334455。02表示messageId，此消息上报数组类型的描述信息；1122334455表示描述信息，长度为5个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=ESlzRF=}。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=ESlzRF=}。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。

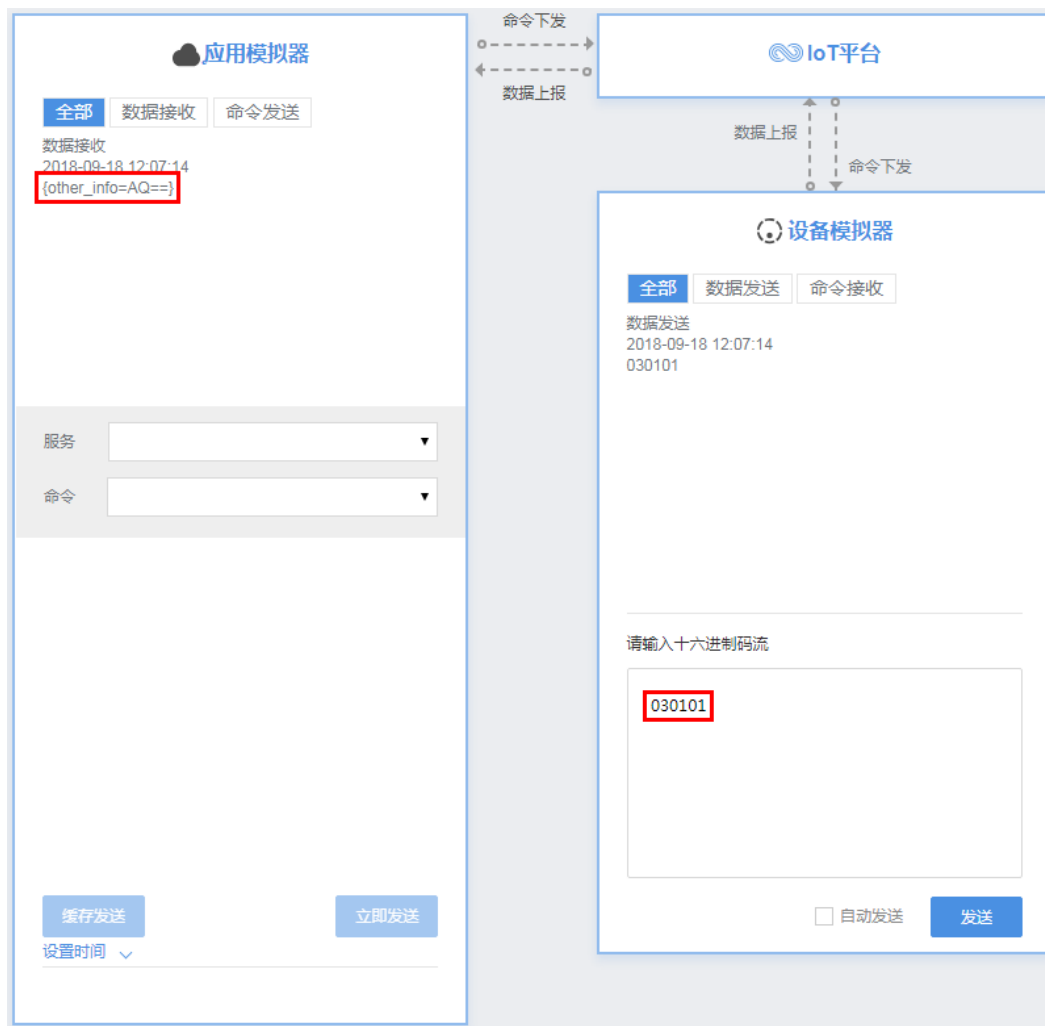


**步骤3** 使用设备模拟器上报可变长度数组类型的描述信息。

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。

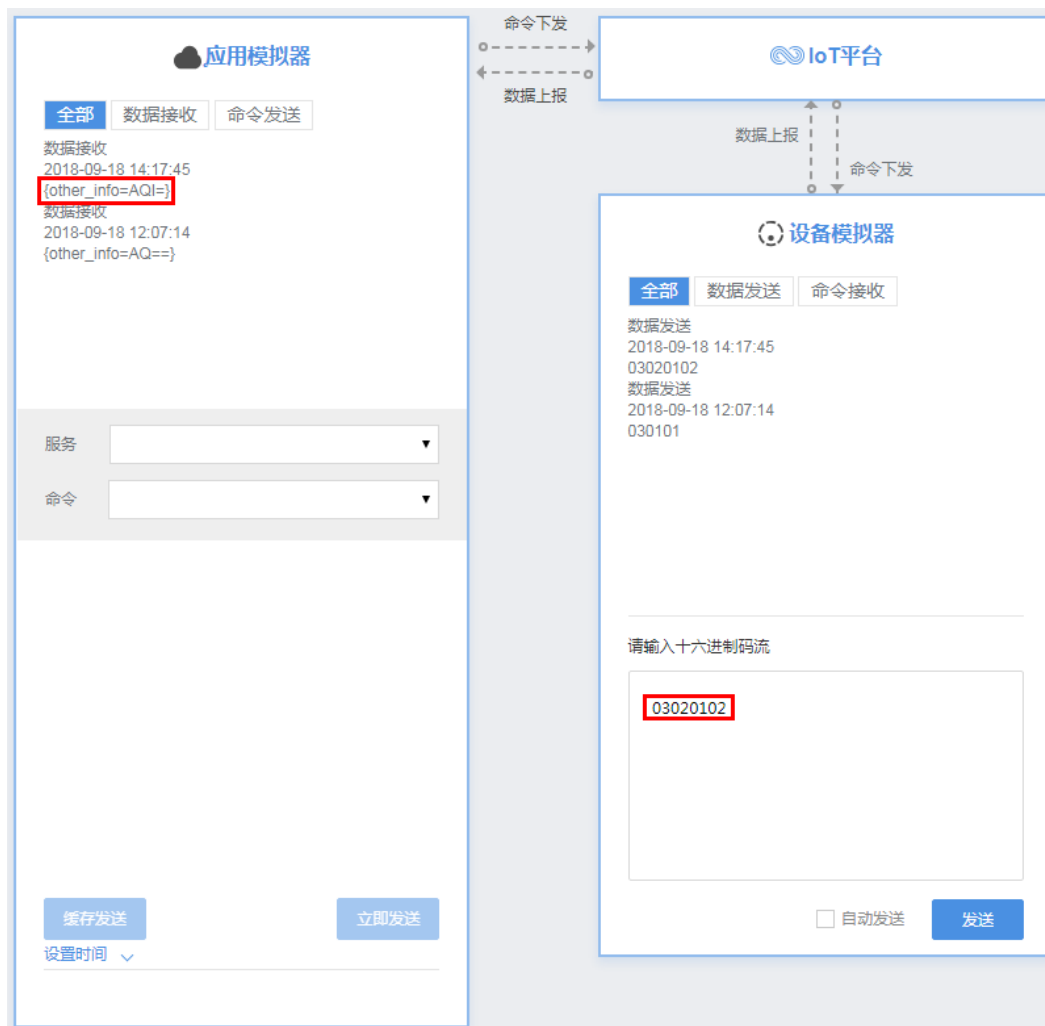
在“应用模拟器”区域查看数据上报的结果：{other\_info=AQ==}。AQ==是01经过base64编码后的值。





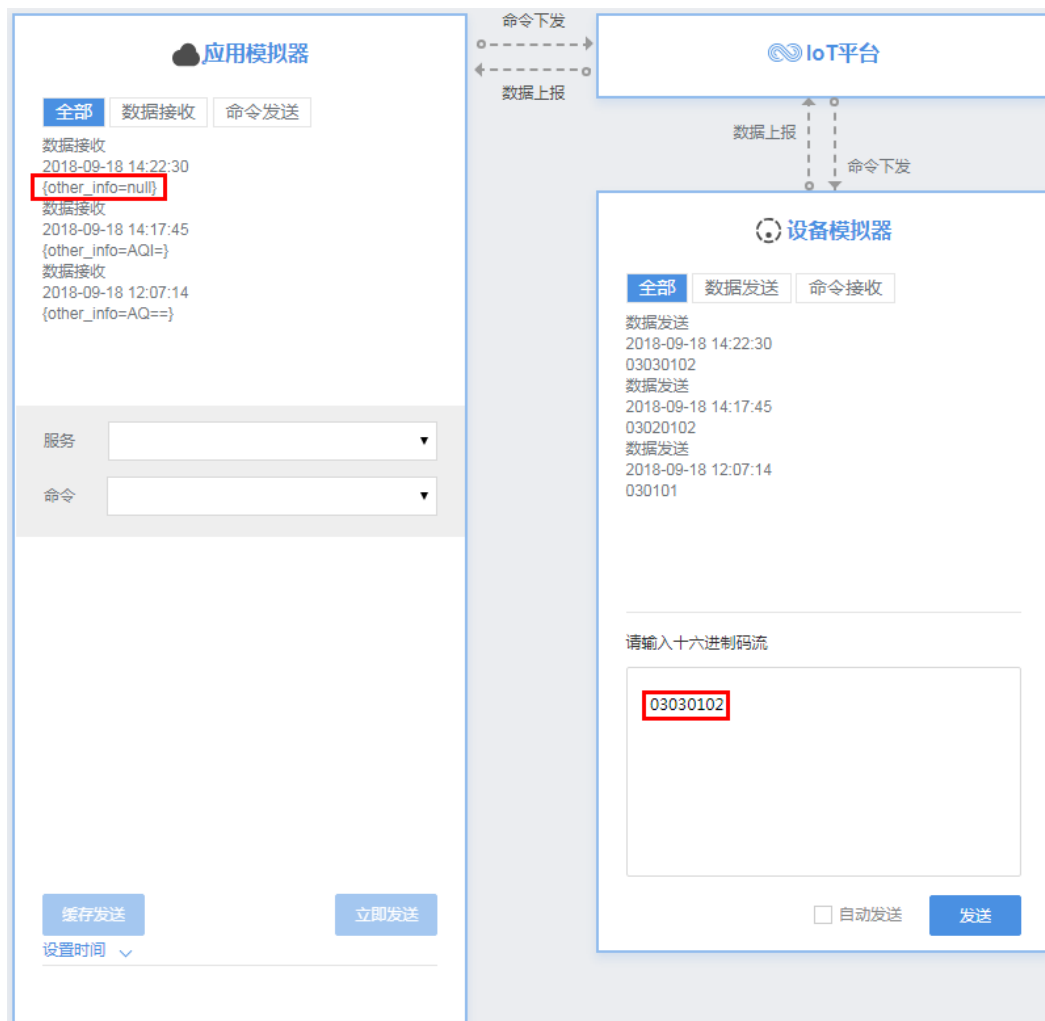
十六进制码流示例：03020102。03表示messageId，此消息上报可变长度数组类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

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。



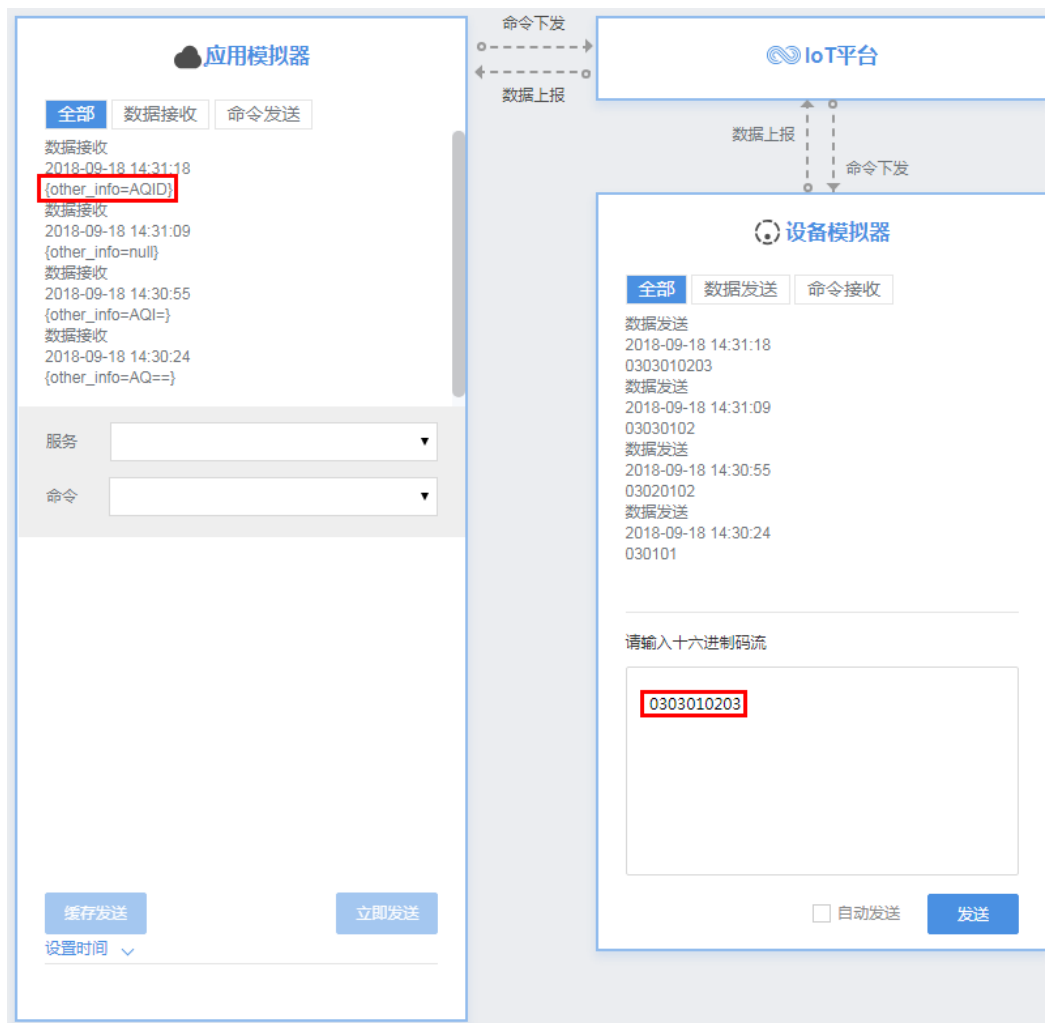
十六进制码流示例：03030102。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

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。



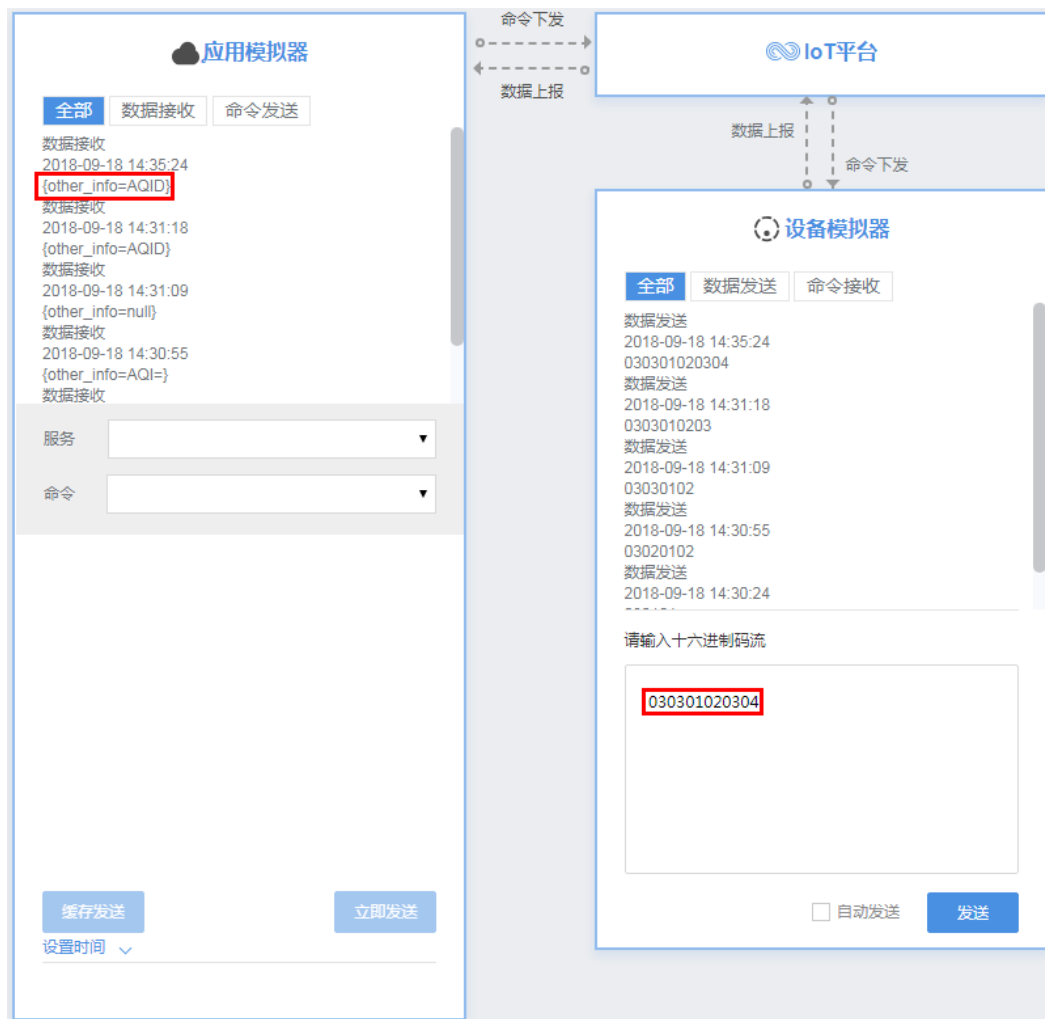
十六进制码流示例：0303010203。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；010203表示描述信息，长度为3个字节。

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，则在码流后用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000000010000000000000000（ $3 \times 8 = 24$ ）。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

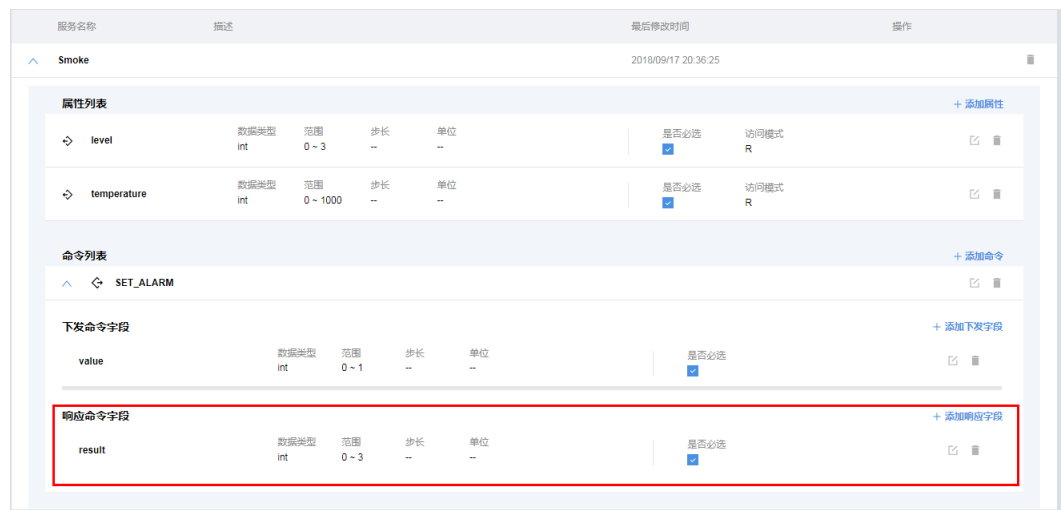
- 当数据类型为数组或可变长度数组时，插件是按照base64进行编解码的：上报数据时，将16进制码流进行base64编码，比如：01编码为“AQ==”；命令下发时，将字符进行base64解码，比如：“AQ==”解码为01。
- 当某字段的数据类型为可变长度数组时，该字段需要关联长度字段，长度字段的数据类型必须为int。
- 针对可变长度数组，命令下发和数据上报的编解码插件开发方式相同。
- 在线开发的编解码插件使用base64进行编码时，是将16进制码流当做数值进行编码。

## 含命令执行结果的编解码插件

如果该烟感设备需要支持支持上报命令执行结果，则按照以下步骤创建消息。

### Profile定义

在烟感产品的开发空间完成Profile定义。



### 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** 配置命令下发消息。

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 ✕

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

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 ✕

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

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3** 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 ×

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 ×

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

result

描述

数据类型

int8u(8位无符号整型) ▼

\* 长度 ?

1

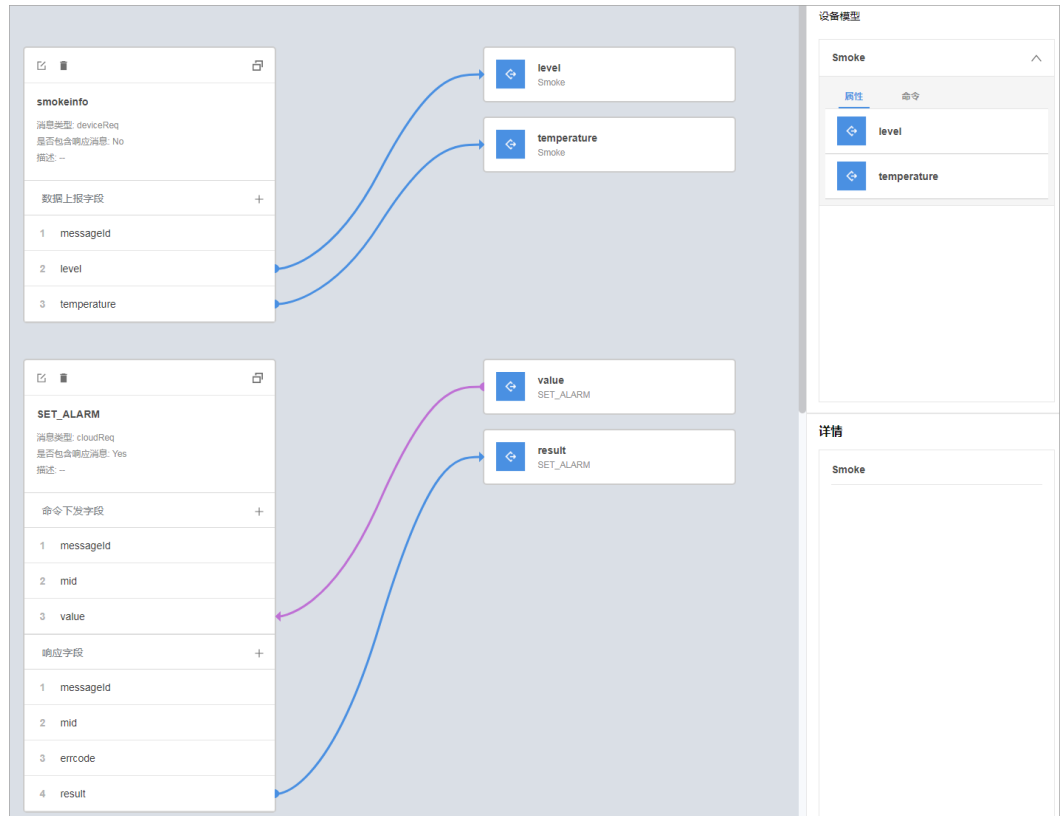
默认值 ?

偏移值 ?

4-5

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。

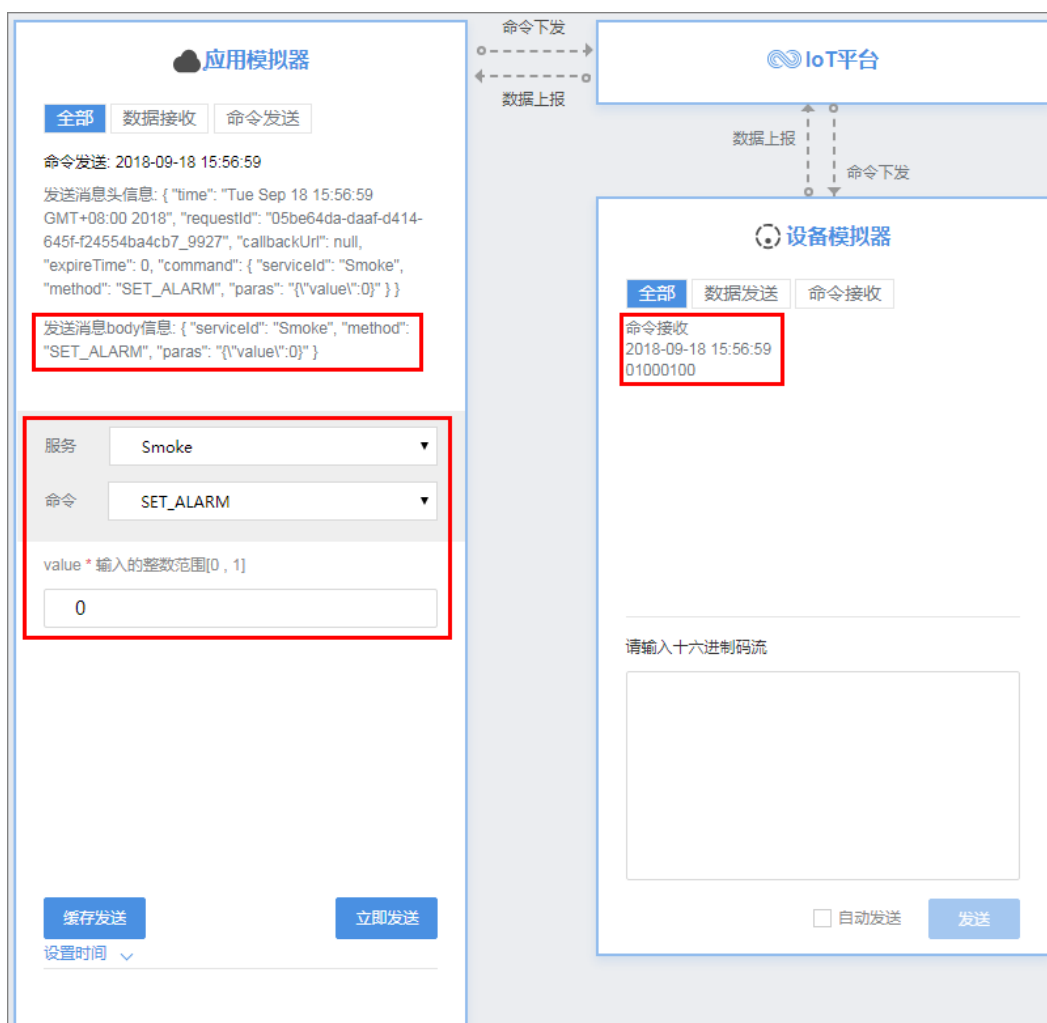


勾选“没有真实的物理设备”，点击“创建”。



**步骤2** 使用应用模拟器进行命令下发：{ "serviceld": "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messageld字段，0001为mid字段，00为value字段。



**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。

状态	命令ID	命令创建时间	命令内容	命令响应
执行成功	963c1d3c37304828a1c97727c23bd268	2018/09/18 15:56:59	{"serviceId":"Smoke","method":"SET_ALARM","paras":{"value":0}}	{"result":0}

---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。

## 4.5.3 离线开发插件（联通用户专用）

非联通用户请查看[设备接入服务](#)。

编解码插件实现二进制消息转JSON格式的功能，Profile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。

为了提高离线开发的集成效率，我们提供了编解码插件的[编解码插件开发样例](#)，建议您基于DEMO工程进行二次开发。

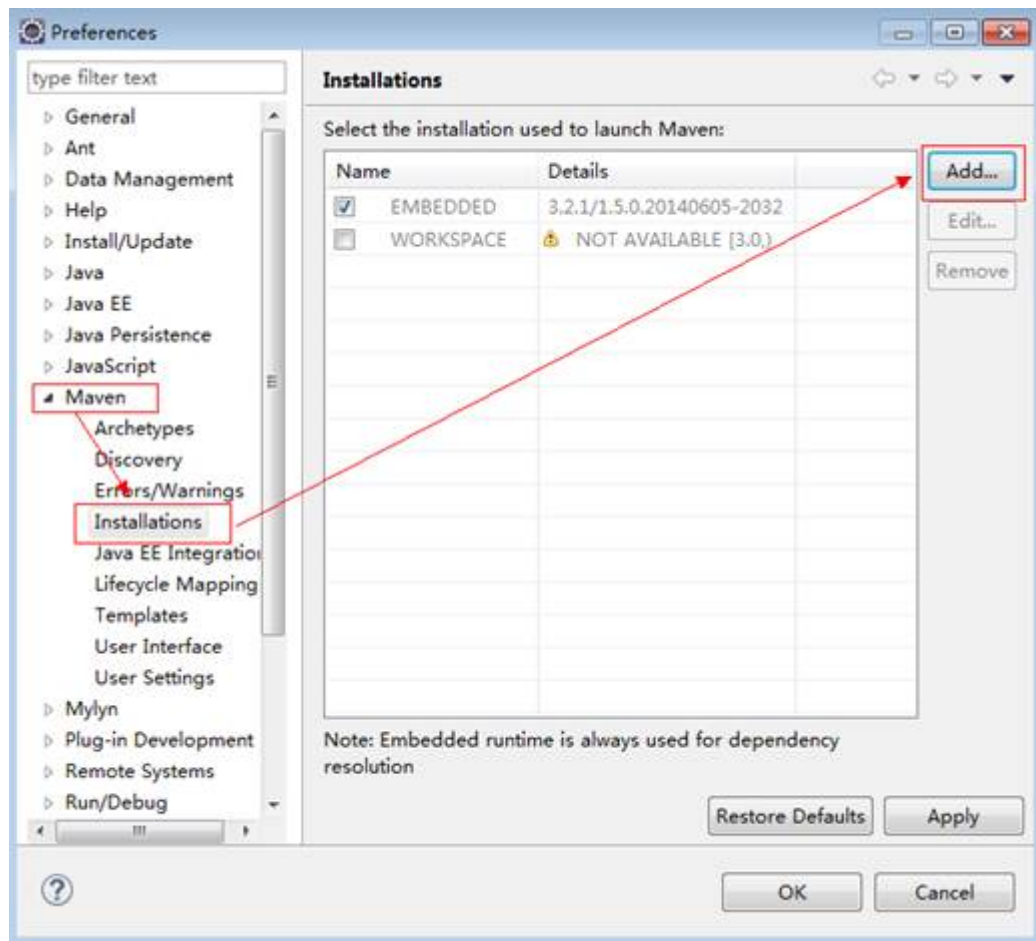
注：由于插件离线开发较为复杂，且耗时比较长，我们推荐[在线开发插件（联通用户专用）](#)。

## 开发环境准备

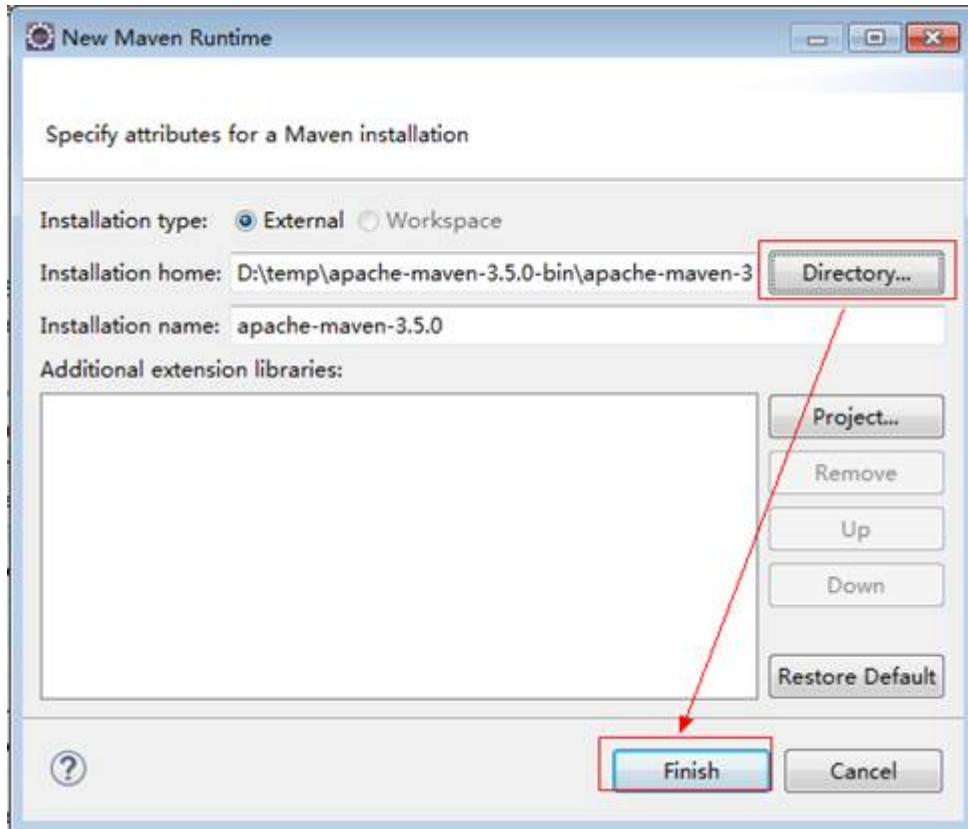
- 前往[官网](#)下载Eclipse安装包，直接解压缩到本地即可使用。
- 前往[官网](#)下载Maven插件包（zip格式），直接解压缩到本地。
- 安装JDK并配置Java的开发环境。

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。

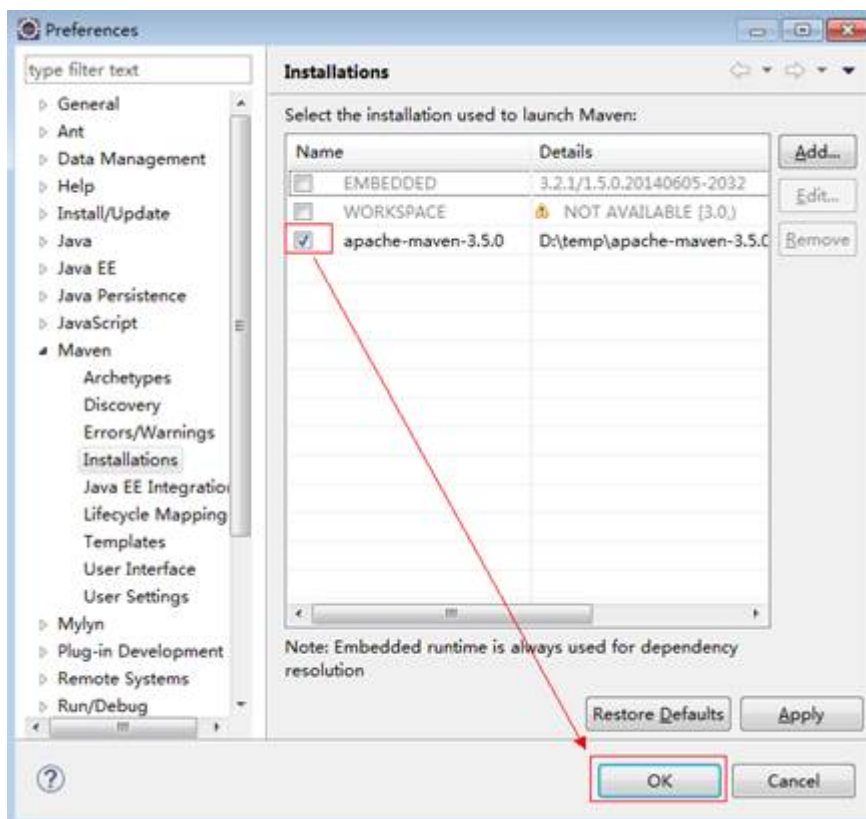
**步骤1** 选择Eclipse菜单“Windows”->“Preferences”，打开Preferences窗口，选择“Maven”->“Installations”->“Add”。



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



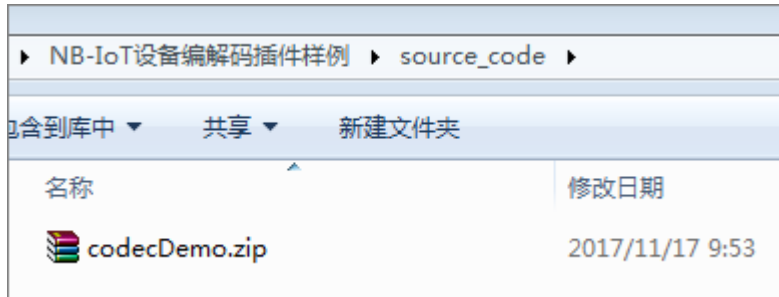
步骤3 选择导入的maven插件，点击“OK”。



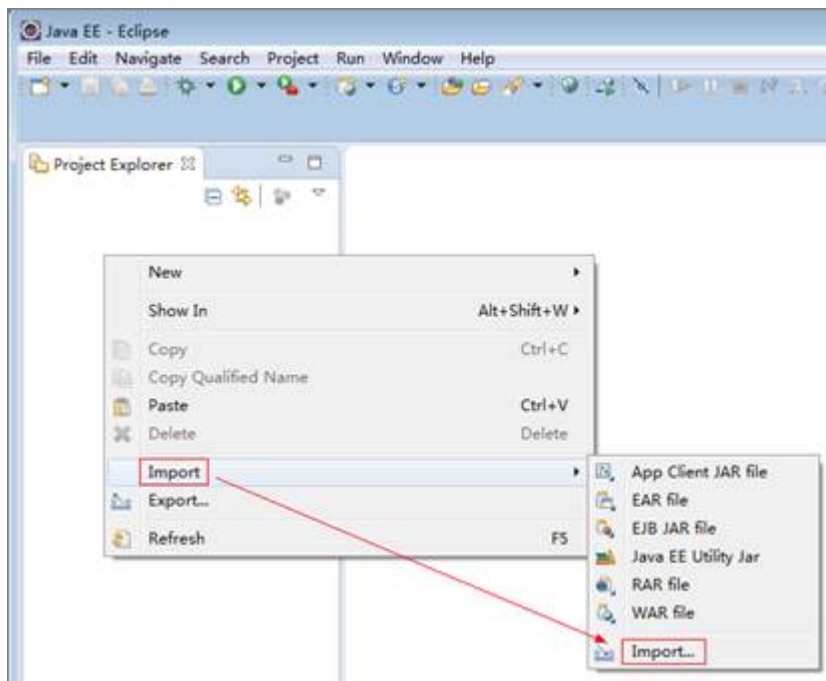
----结束

## 导入编解码插件 Demo 工程

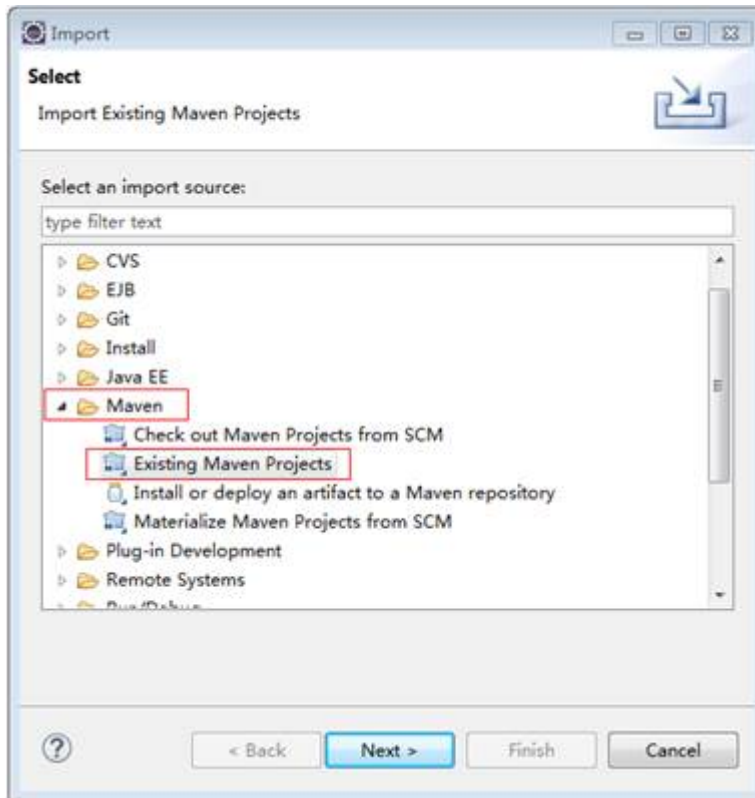
**步骤1** 下载编解码插件DEMO工程，在“source\_code”文件夹中获取“codecDemo.zip”，将其解压到本地。



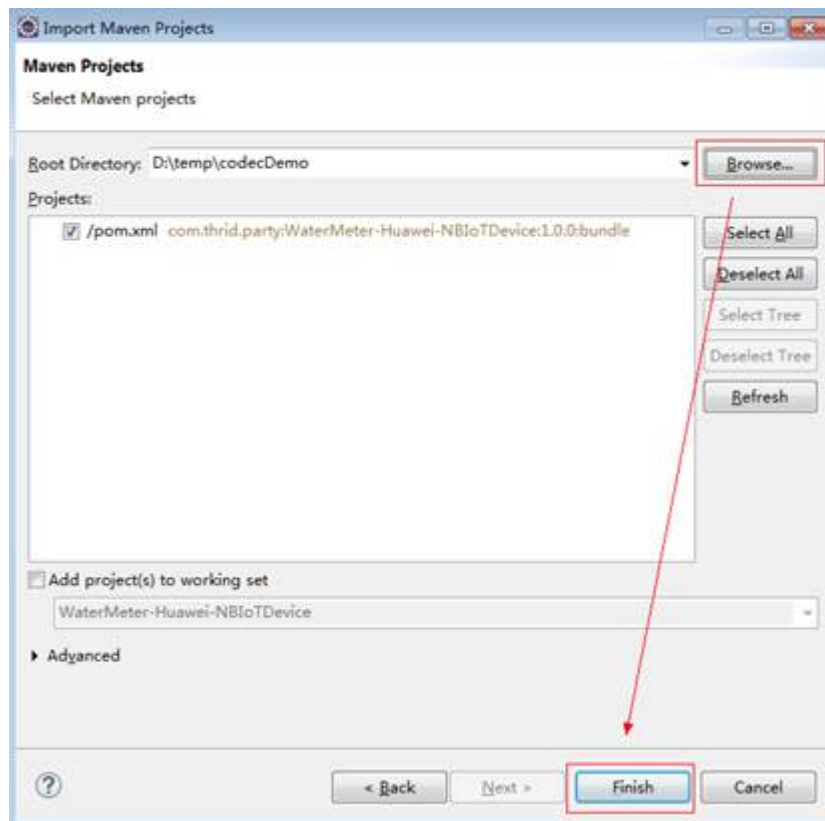
**步骤2** 打开Eclipse，右击Eclipse左侧“Project Explorer”空白处，选择“Import > Import...”。



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



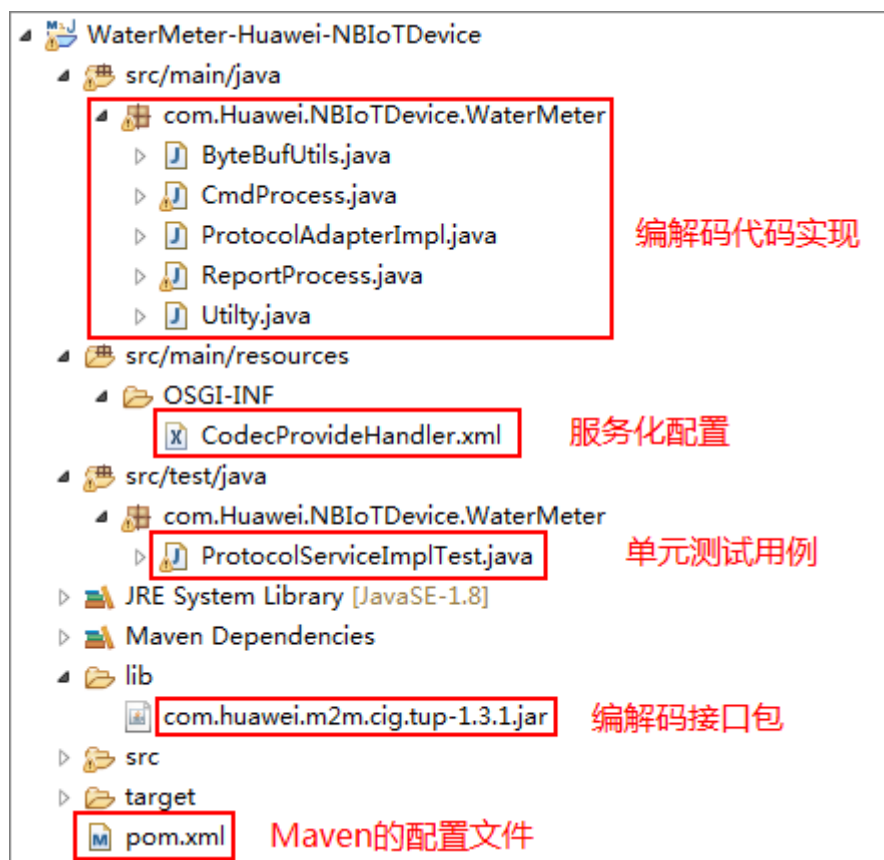
**步骤4** 点击“Browse”，选择步骤1解压获得的“codecDemo”文件夹，勾选“/pom.xml”，点击“Finish”。



----结束

## 实现样例讲解

导入的编解码插件Demo工程结构如下图所示。



本工程是一个Maven工程，您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。

### 步骤1 修改Maven的配置文件

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<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-NB IoT Device -->
<artifactId>WaterMeter-Huawei-NB IoT Device</artifactId>
<version>1.0.0</version>
<!-- 请检查这里的值为bundle，不能为jar -->
<packaging>bundle</packaging>

.....

<dependencies>
.....
<!-- Huawei提供的编解码接口，必须引入 -->
<!-- systemPath请替换成你本地的目录 \codecDemo\lib\com.huawei.m2m.cig.tup-1.3.1.jar -->
<dependency>
<groupId>com.huawei</groupId>
<artifactId>protocal-jar</artifactId>
<version>1.3.1</version>
<scope>system</scope>
<systemPath>${basedir}/lib/com.huawei.m2m.cig.tup-1.3.1.jar</systemPath>
</dependency>
```

```
.....
</dependencies>
</build>
<plugins>
  <!-- OSGI规范打包配置 -->
  <plugin>
    <configuration>
      <instructions>
        <!-- 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-NBLoTDevice -->
        <Bundle-SymbolicName>WaterMeter-Huawei-NBLoTDevice</Bundle-SymbolicName>
      </instructions>
    </configuration>
  </plugin>
</plugins>
</build>
</project>
```

**步骤2** 在ProtocolAdapterImpl.java中，修改厂商ID（MANU\_FACTURERID）和设备型号（MODEL）的取值。物联网平台通过厂商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 = "NBLoTDevice";
```

**步骤3** 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.NBLoTDevice.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",
  "serviceld":"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 (Uility.getInstance().isValidofMid(mid)) {
          byte[] byteMid = new byte[2];
          byteMid = Uility.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;
    }
}
```

#### 步骤4 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.NBLoTDevice.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;

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

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

    //serviceld=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":[{"serviceld":"Brightness",
                "serviceData":{"brightness":50},
                {
                    "serviceld":"Electricity",
                    "serviceData":{"voltage":218.9,"current":800,"frequency":50.1,"powerfactor":0.98},
                    {
                        "serviceld":"Temperature",

```

```
        "serviceData":{"temperature":25},
        ]
    }
    */
    if (binaryData[2] == bDeviceReq) {
        msgType = "deviceReq";
        hasMore = binaryData[3];

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

        //servicId=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;

        //servicId=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 = Uility.getInstance().bytes2Int(binaryData, 5, 2);
            if (Uility.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();

            //servicId=Brightness 数据组装
            ObjectNode brightNode = mapper.createObjectNode();
            brightNode.put("servicId", "Brightness");
            ObjectNode brightData = mapper.createObjectNode();
            brightData.put("brightness", this.brightness);
            brightNode.put("serviceData", brightData);
            arrynode.add(brightNode);
            //servicId=Electricity 数据组装
```

```
ObjectNode electricityNode = mapper.createObjectNode();
electricityNode.put("serviceld", "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);
//serviceld=Temperature 数据组装
ObjectNode temperatureNode = mapper.createObjectNode();
temperatureNode.put("serviceld", "Temperature");
ObjectNode temperatureData = mapper.createObjectNode();
temperatureData.put("temperature", this.temperature);
temperatureNode.put("serviceData", temperatureData);
arrynode.add(temperatureNode);

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

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

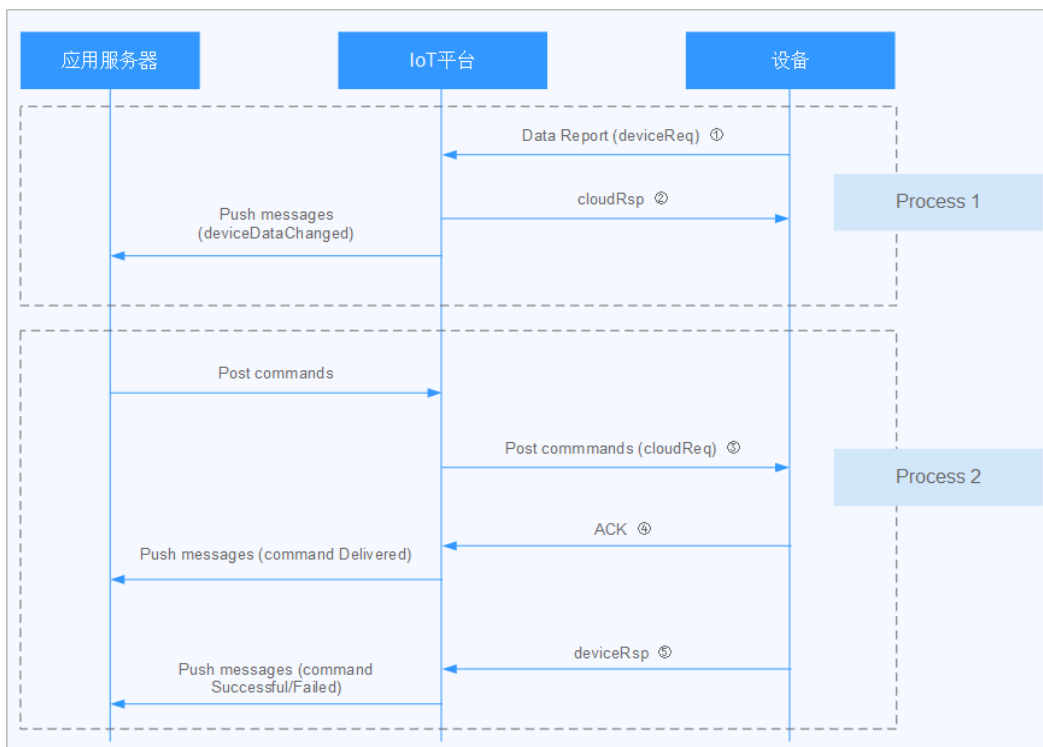
root.put("data", arrynode);

} 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;
}
}
```

---结束

## 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。



设备的上行报文有两种情况需要插件处理（消息④是模组回复的协议ACK，无需插件处理）：

- 设备上报数据（对应图中的消息①）

字段名	类型	是否必填	参数描述
identifier	String	否	设备在应用协议里的标识，物联网平台通过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。
msgType	String	是	固定值"deviceReq"，表示设备上报数据。
hasMore	Int	否	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，物联网平台以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，物联网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，则物联网平台按照hasMore字段为0处理。
data	ArrayNode	是	设备上报数据的内容。

表 4-1 ArrayNode 定义

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

示例：

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

- 设备对平台命令的应答（对应图中的消息⑤）

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，物联网平台通过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

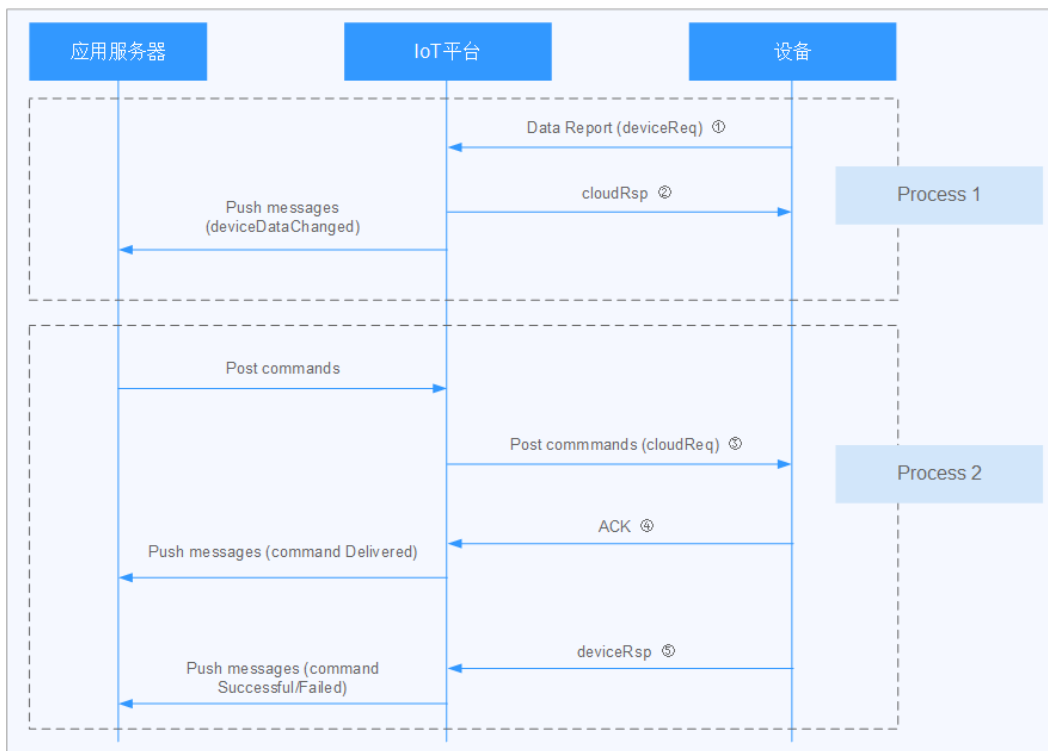
字段名	类型	参数描述	是否必填
mid	Int	2字节无符号的命令id。在设备需要返回命令执行结果（deviceRsp）时，用于将命令执行结果（deviceRsp）与对应的命令进行关联。  物联网平台在通过encode接口下发命令时，把物联网平台分配的mid放入码流，和命令一起下发给设备；设备在上报命令执行结果（deviceRsp）时，再将此mid返回给物联网平台。否则物联网平台无法将下发命令和命令执行结果（deviceRsp）进行关联，也就无法根据命令执行结果（deviceRsp）更新命令下发的状态（成功或失败）。	是
errcode	Int	请求处理的结果码，物联网平台根据该参数判断命令下发的状态。 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>注：</b> body体不是数组。	否

**示例：**

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

## encode 接口说明

encode接口的入参是JSON格式的数据，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：

- 平台对设备上报数据的应答（对应图中的消息②）

表 4-2 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，物联网平台通过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，物联网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	表示平台是否还有后续消息下发，0表示没有，1表示有。 后续消息是指，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

**注：**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，设备侧不需要物联网平台给予响应。

**示例：**

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

- 平台命令下发（对应图中的消息③）

**表 4-3** 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，物联网平台通过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
mid	Int	2字节无符号的命令id，由物联网平台内部分配（范围1-65535）。 物联网平台在通过encode接口下发命令时，把物联网平台分配的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返回物联网平台。否则物联网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

**示例：**



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

## getManufacturerId 接口说明

返回厂商ID字符串。物联网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 "TestUtf8Manuld";
}
```

## getModel 接口说明

返回设备型号字符串。物联网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";
}
```

## 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;

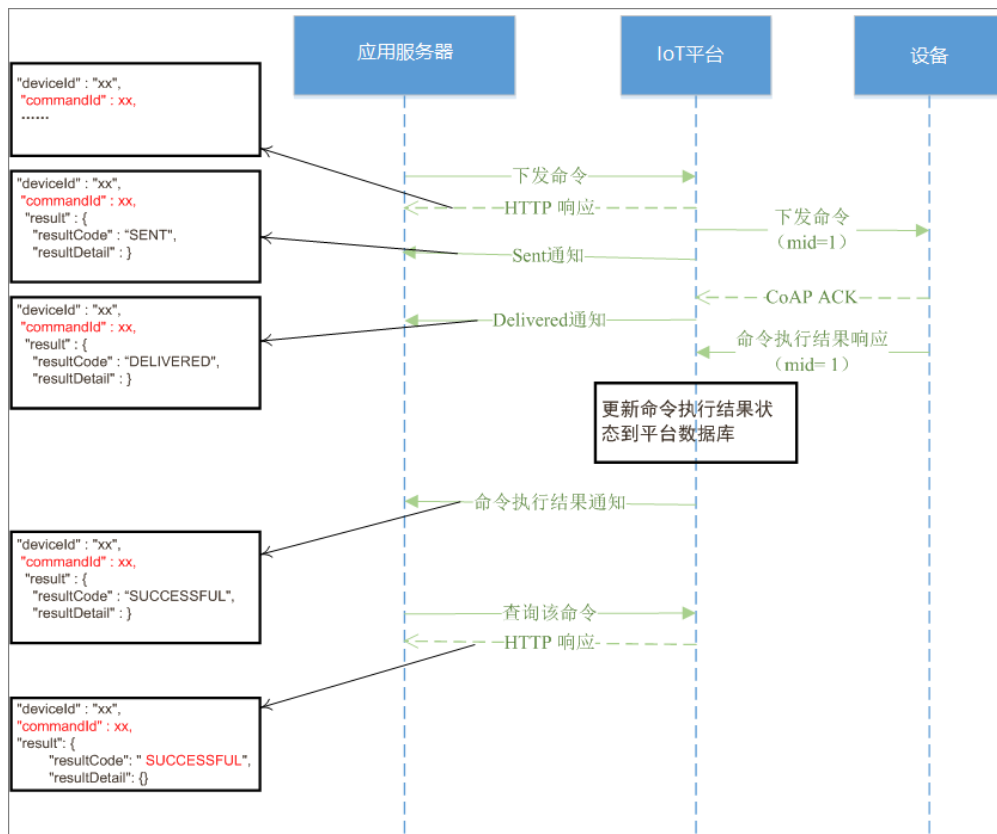
    @Override
    public ObjectNode decode(finalbyte[] binaryData) throws Exception {
        if (binaryData == null) {
            status = "Failed";
            return null;
        }
        ObjectNode node;
        ...;
        status = "Success";// 线程不安全
        return node;
    }
}
```

- 正确示例：直接使用入参编解码，编解码库不做业务处理。

### mid字段的解释

物联网平台是依次进行命令下发的，但物联网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。在物联网平台，是否实现mid，消息流程也有所不同：

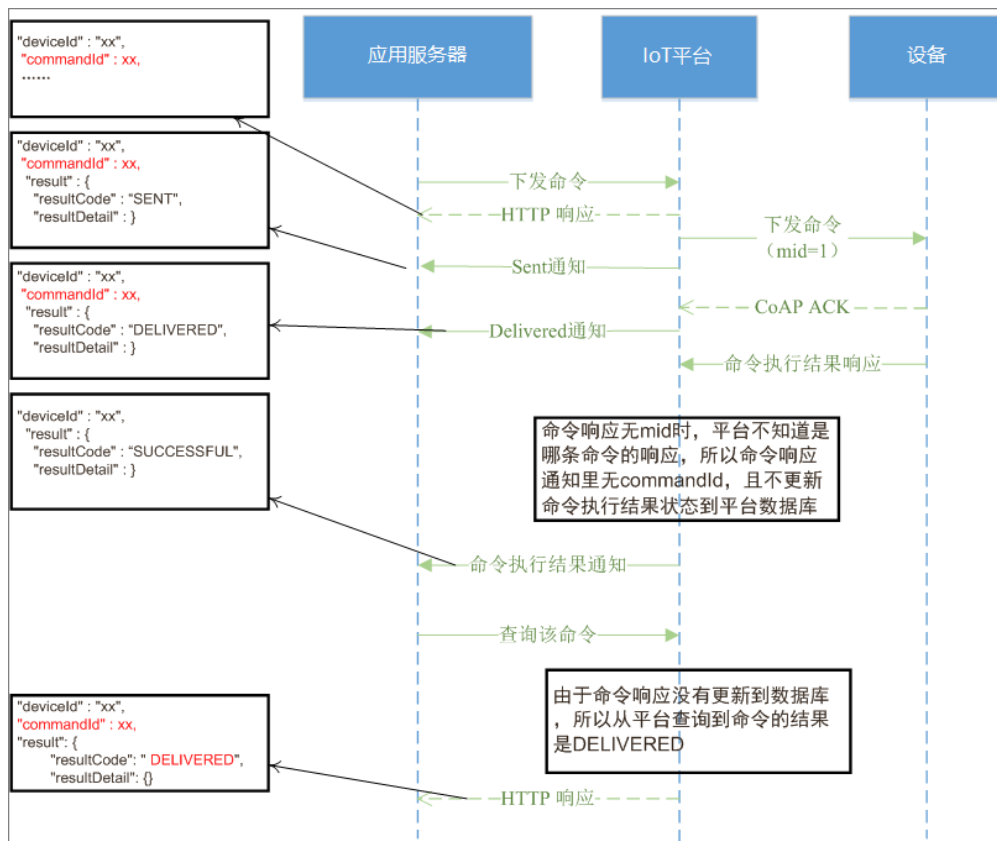
● 实现mid



若实现了mid，并且命令执行结果已上报成功，则：

- 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；
- 平台推送给应用服务器的命令执行结果通知中携带commandId；
- 应用服务器查询会得到该命令的状态为SUCCESSFUL/FAILED。

● 不实现mid



若不实现mid，并且命令执行结果已上报成功，则：

- 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；
- 平台推送给应用服务器的命令执行结果通知中不携带commandId；
- 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。

如果不实现mid，则应用服务器不能在物联网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
{
    ...
}
```

## 编解码插件的输入/输出格式样例

假定某款水表支持的服务定义如下：

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": [
    {
      "serviceld": "Meter",
      "serviceData": {
        "currentReading": "46.3",
        "signalStrength": 16,
        "dailyActivityTime": 5706
      },
      "eventTime": "20160503T121540Z"
    },
    {
      "serviceld": "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
}

```

假定某款水表支持的命令定义如下：

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_RSP	-	-	-
-	-	-	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",
  "servicId": "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
  }
}

```

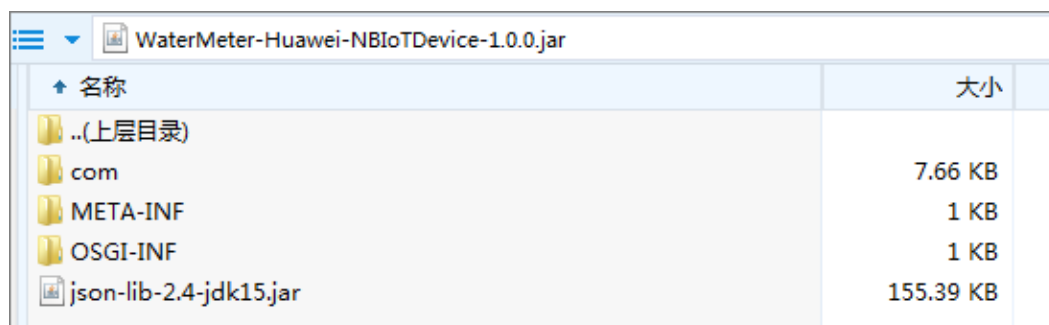
## 编解码插件打包

插件变成完成后，需要使用Maven打包成jar包，并制作成插件包。

## Maven打包

- 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-NBLoTDevice-version.jar。



- 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。



- 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：

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

表 4-4 “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

字段名	字段描述	是否必填
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
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，本插件包运行的物联网平台的操作系统，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。 注：bundle就是压缩包中的jar包，只需要写一个bundle。	是

表 4-5 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":"NBloTDevice",
"description":"codec",
"platform":"linux",
"packageType":"CIGPlugin",
"date":"2017-02-06 12:16:59",
"ignoreList":[],
"bundles":[
  {
    "bundleName": "WaterMeter-Huawei-NBloTDevice",
    "bundleVersion": "1.0.0",
    "priority":5,
    "fileName": "WaterMeter-Huawei-NBloTDevice-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”）。

注：“package.zip”中不能包含“package”这层目录。

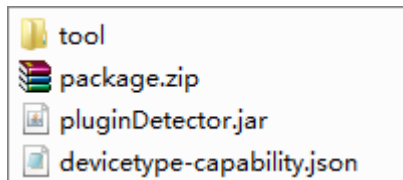
----结束

## 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** 获取[编解码插件检测工具](#)。

**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。

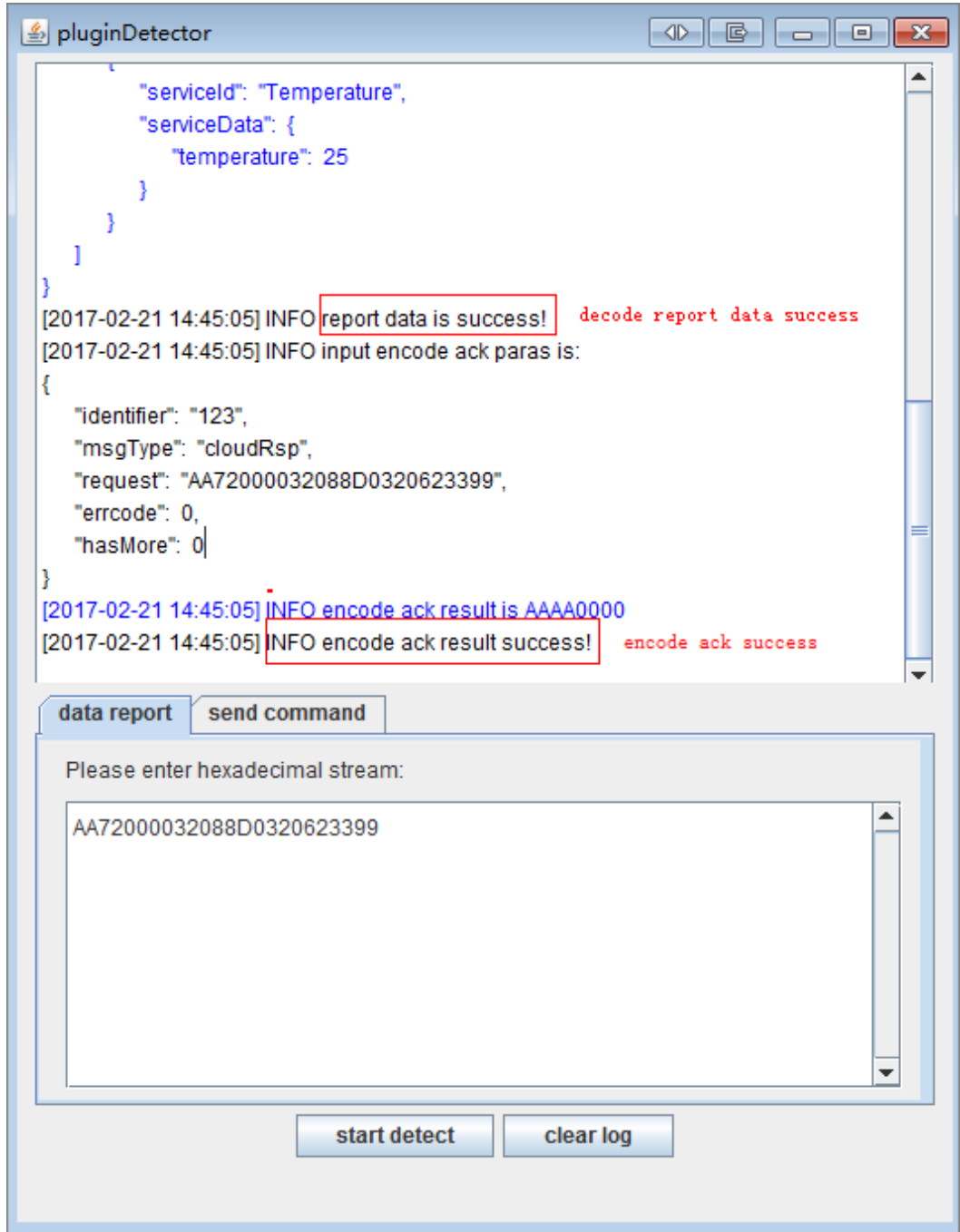


**步骤3** 获取设备数据上报的码流，并在检测工具的“data report”页签，将码流以十六进制格式输入，例如：AA72000032088D0320623399。

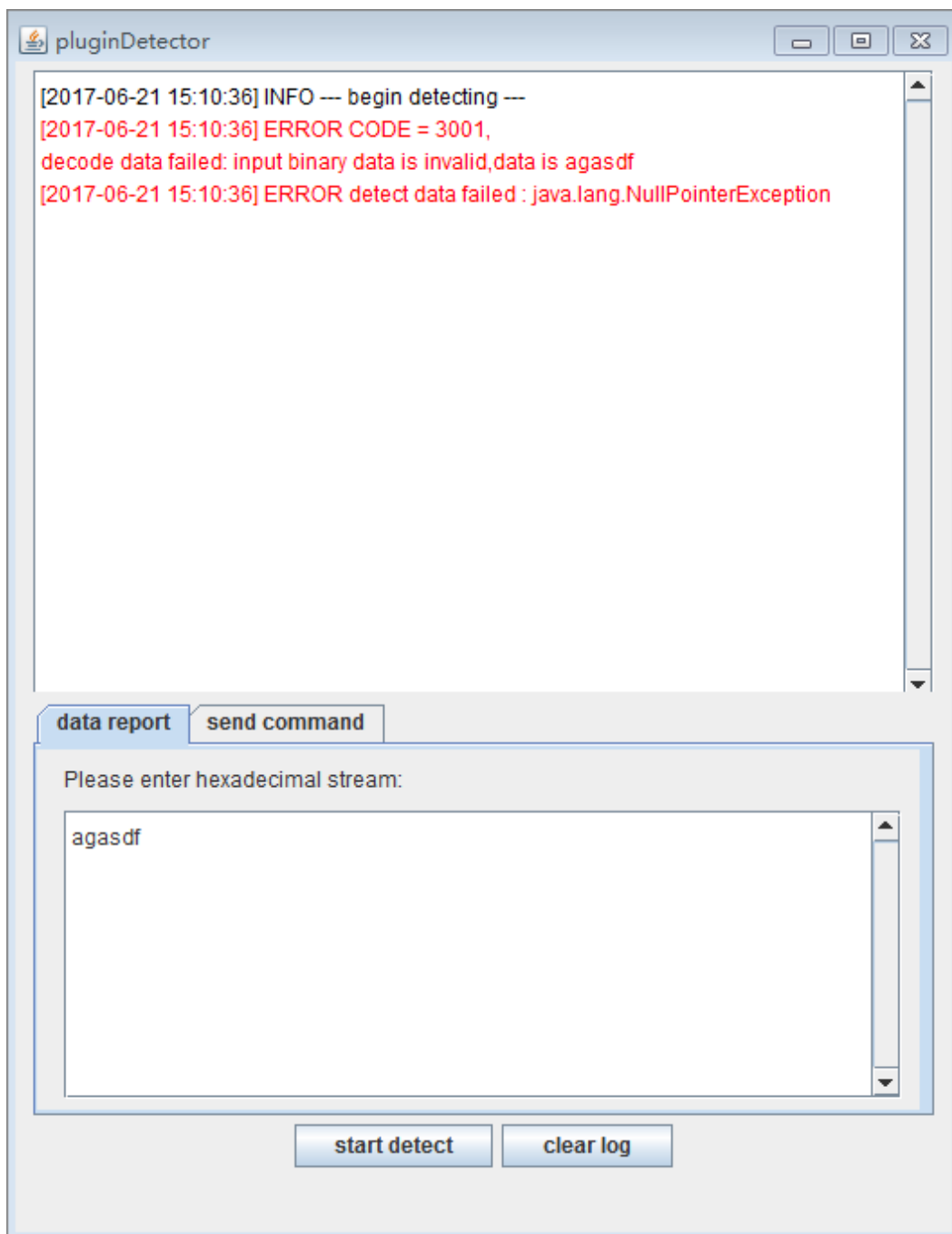
**步骤4** 点击检测工具的“start detect”，查看解码后的json数据。

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。



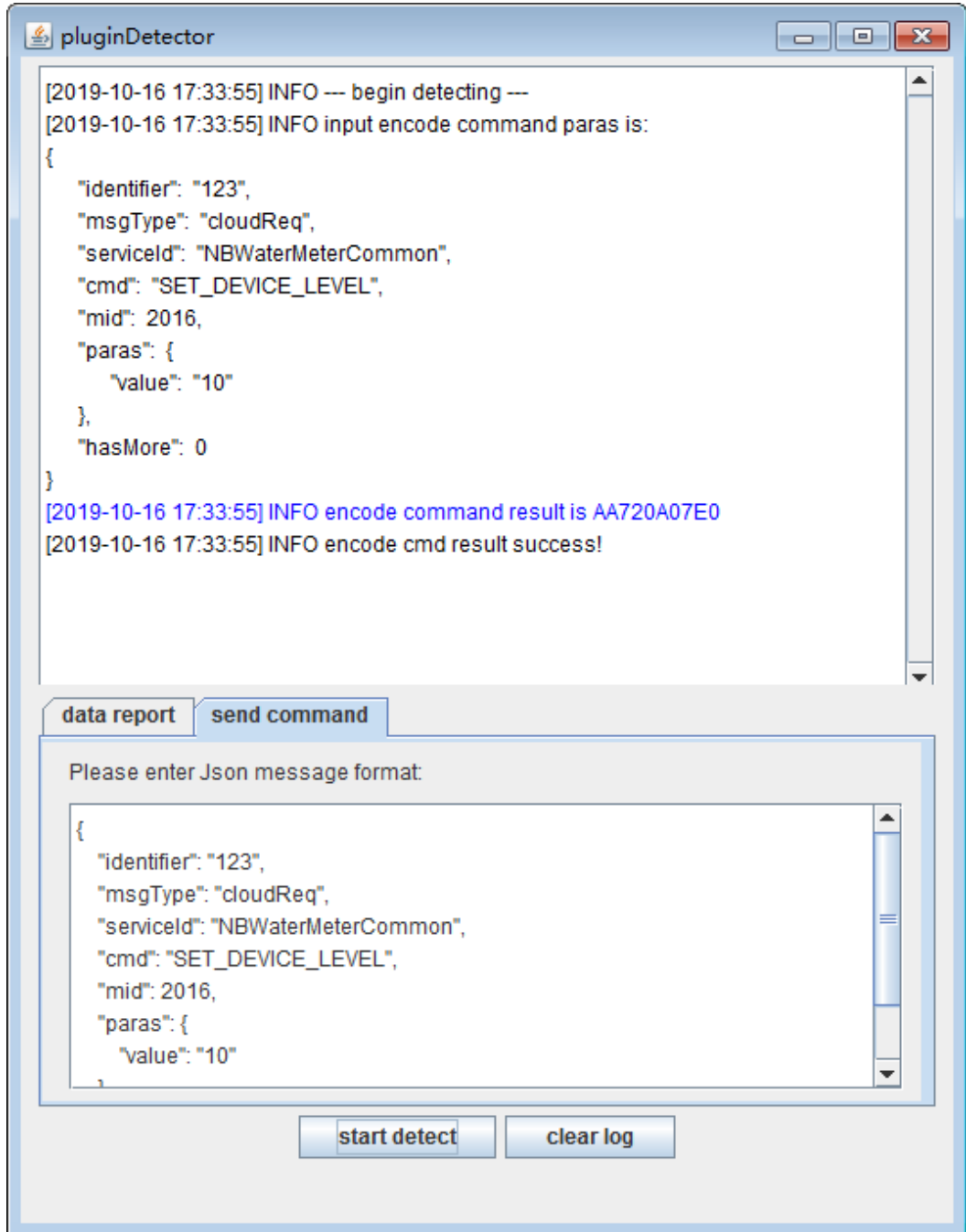
**步骤5** 当解码成功后，检测工具会继续调用编解码插件包的encode方法，对应答消息进行编码。

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。



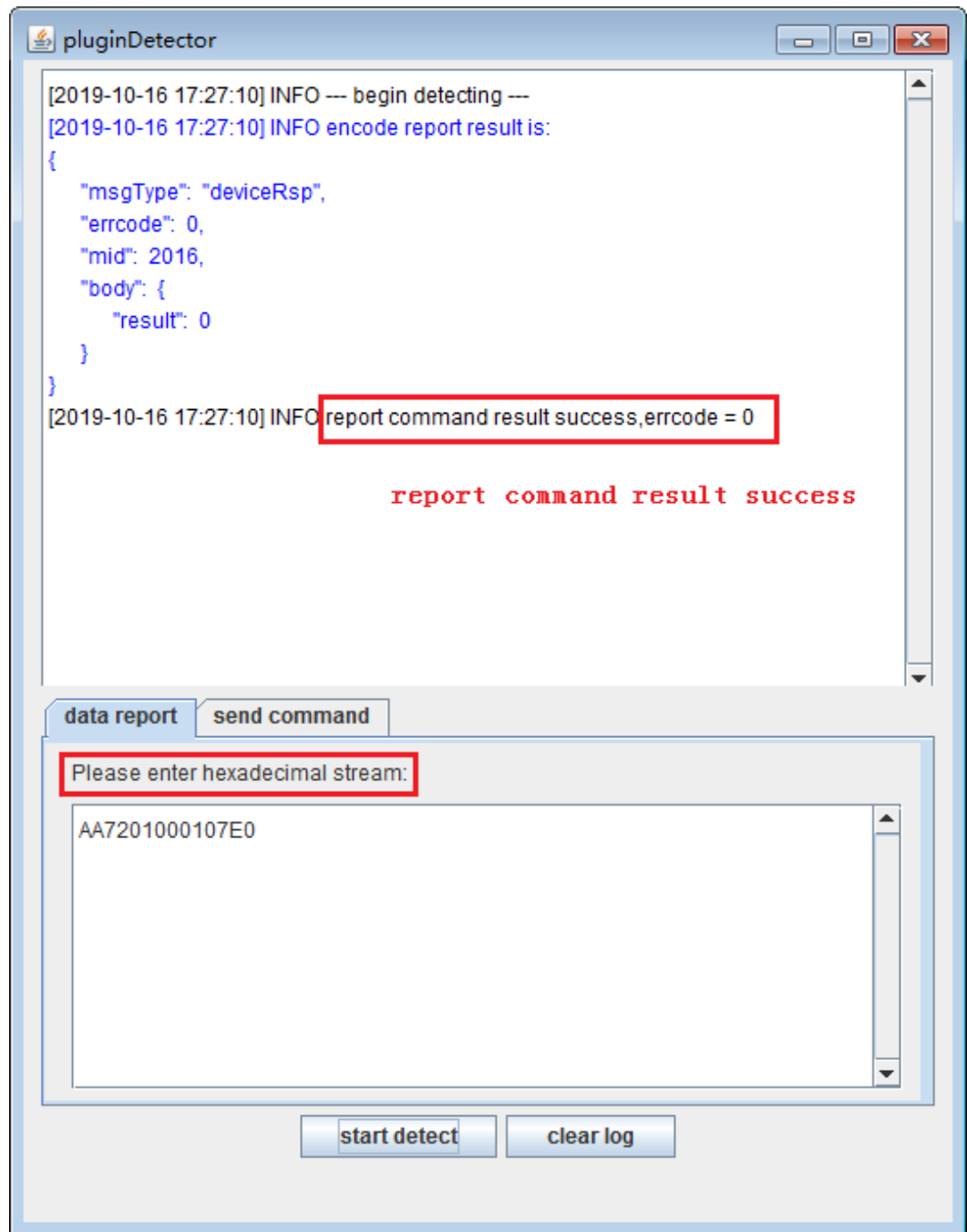
### 命令示例:

```
{
  "identifier": "123",
  "msgType": "cloudReq",
  "serviceld": "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。



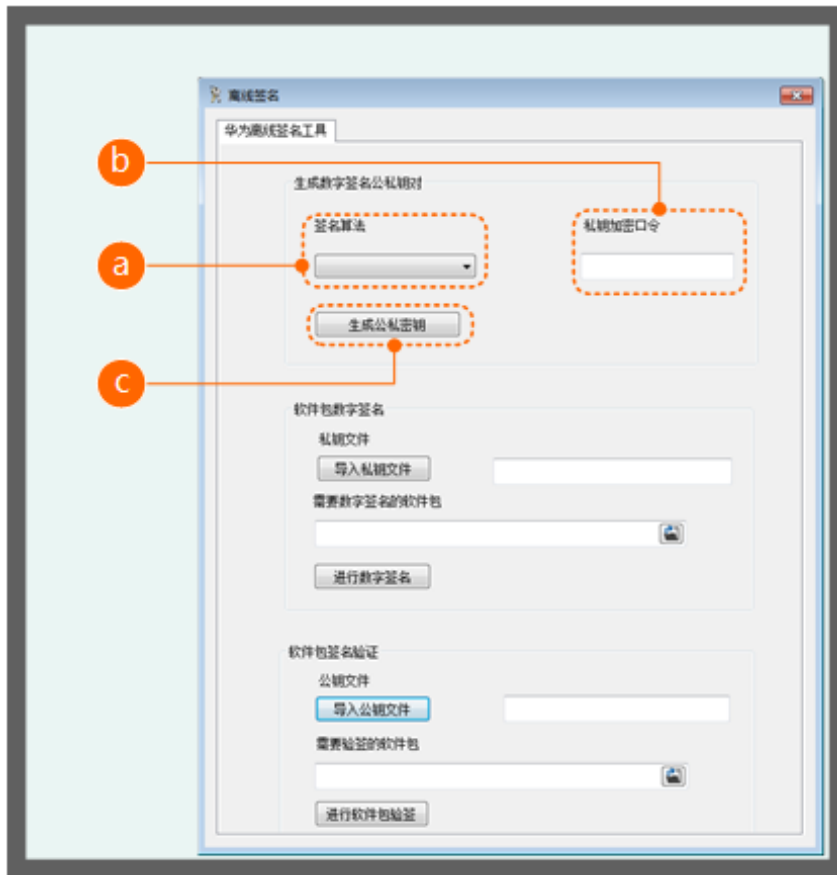
---结束

## 编解码插件包离线签名

当编解码插件开发完后，在安装到物联网平台之前，您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。

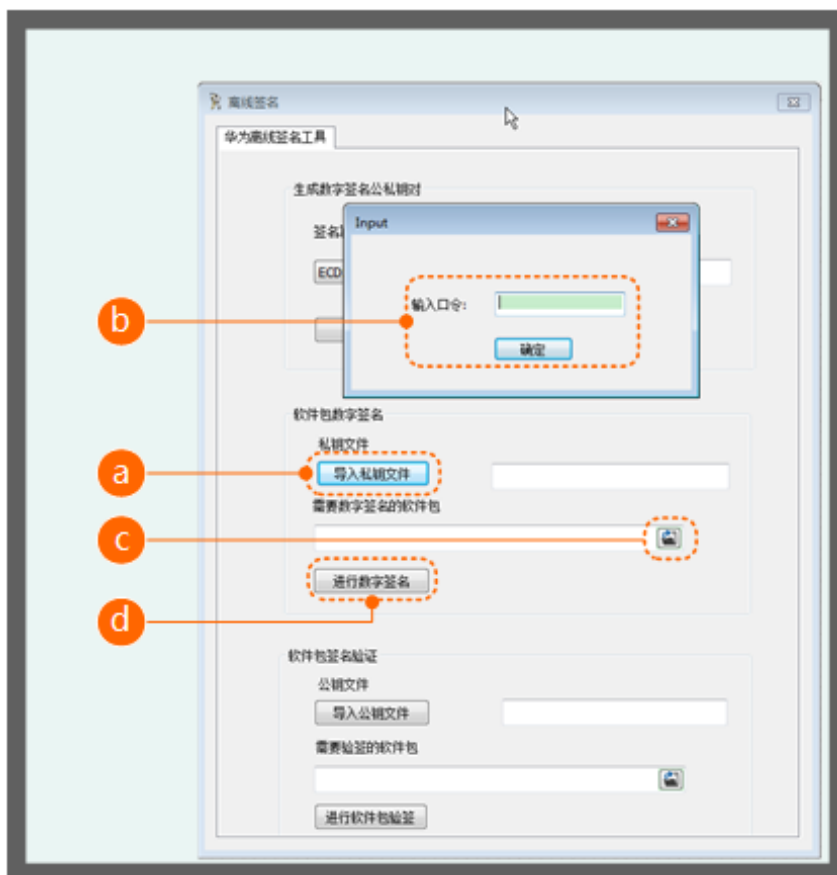
**步骤1** 登录“设备管理服务控制台”，选择“系统管理 > 工具”，单击 [↓](#)，完成工具下载。


**步骤2** 解压“signtool.zip”，双击“signtool.exe”，生成数字签名公私钥对。



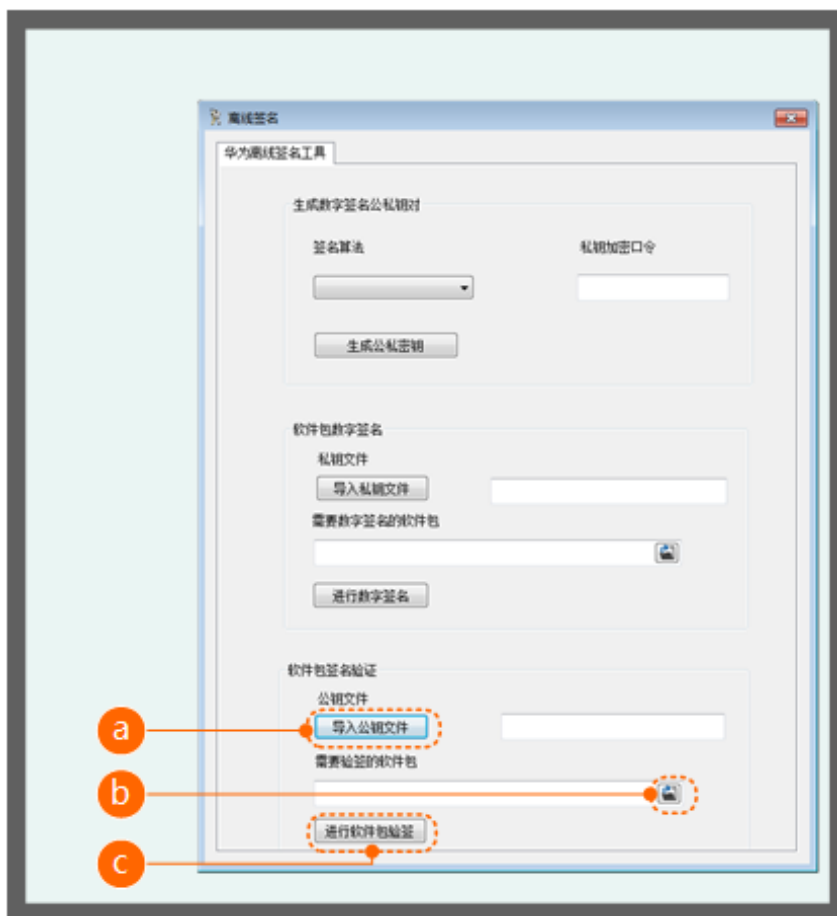
1. 根据需要选择“签名算法”，当前提供两种签名算法：。
  - ECDSA\_256K1+SHA256
  - RSA2048+SHA256
2. 输入“私钥加密口令”，口令复杂度需要满足如下条件：
  - 口令长度至少6个字符
  - 必须包括A-Z、a-z、0-9、:~`@#\$\$%^&\*()-\_+=|?/<>[]{}.,;'"中至少两种字符组合。
3. 单击“生成公私密钥”，在弹出框中选择保存密钥的路径，单击“确定”，弹出“Success!”提示框表示密钥生成完成。  
工具将生成公私密钥共2个文件：
  - 公钥文件：“public.pem”
  - 私钥文件：“private.pem”。


**步骤3** 进行软件包数字签名。



1. 单击“导入私钥文件”，导入生成的私钥文件“private.pem”。
2. 在弹出的窗口中输入“私钥加密口令”。  
口令输入正确，则状态栏显示私钥文件路径；口令输入错误，则状态栏显示“私钥导入失败”。
3. 单击  选择待签名软件路径，单击“进行数字签名”按钮，。  
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。

**步骤4** 验证软件包签名。



1. 单击“导入公钥文件”，导入公钥文件“public.pem”。
  2. 单击选择待验证的软件包路径，单击“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。
- 注：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

## 4.5.4 下载和上传插件（联通用户专用）

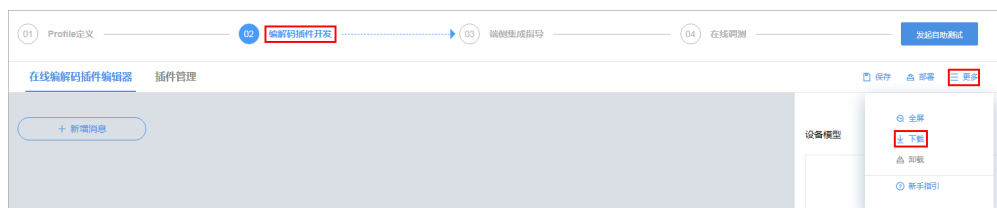
非联通用户请查看[设备接入服务](#)。

在线开发完成编解码插件后，可以将插件下载到本地。本地的插件也可以上传到其他任意的物联网平台上。

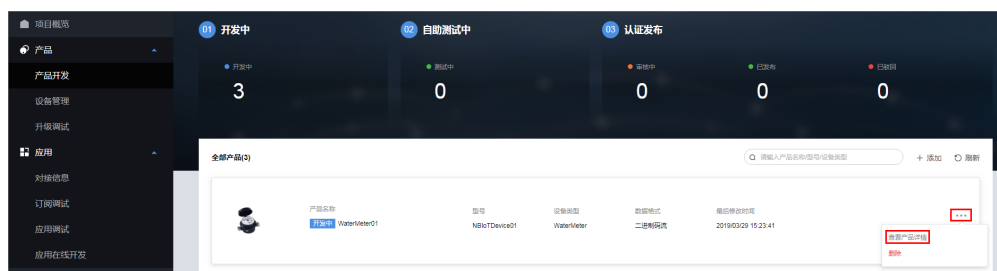
### 下载编解码插件

**步骤1** 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”界面，选择“更多 > 下载”。



- 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。

注：密码长度至少6个字符，必须为字母和数字的组合。





- 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。

注：下载的压缩包里的私钥文件即为上传的私钥。

请选择插件签名方式

生成新的公私钥签名  使用已有私钥签名

\*上传私钥

private.pem 选择文件

使用私钥给插件包签名

\*私钥密码

下载签名插件包需要设置签名密码。

下载 取消

----结束

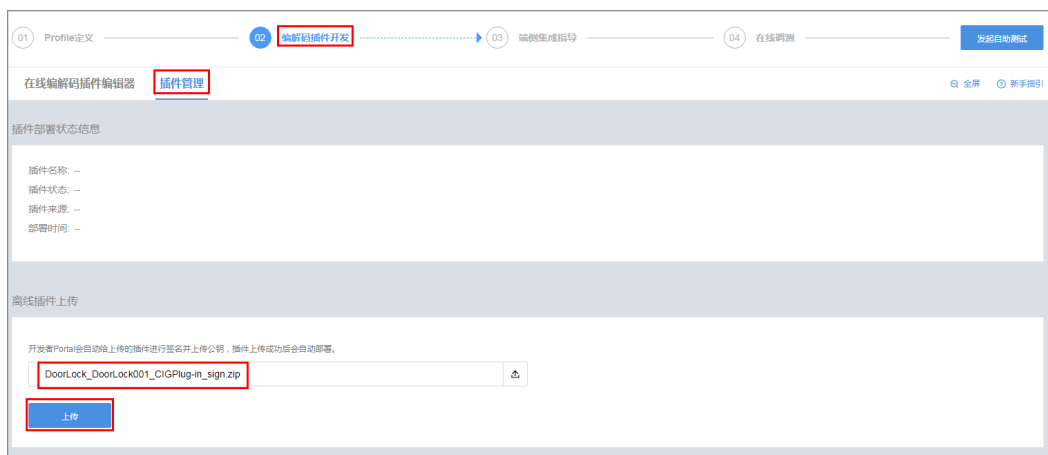
## 上传编解码插件

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。

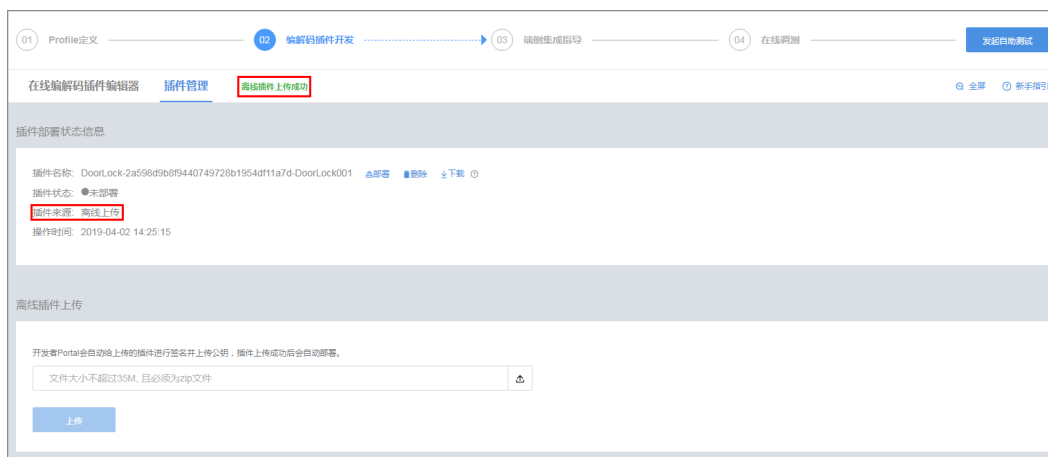
**步骤1** 在“编解码插件开发”界面，选择“插件管理”。

**步骤2** 选择需要上传的插件包，点击“上传”。

注：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

## 4.6 调测产品（联通用户专用）

非联通用户请查看[设备接入服务](#)。

当Profile和编解码插件开发完成后，应用服务器就可以通过物联网平台接收设备上报的数据以及向设备下发命令。

开发中心提供了产品在线调测的功能，您可以根据自己的业务场景，在开发真实应用和真实设备之前，使用应用模拟器和设备模拟器对数据上报和命令下发等场景进行调测；也可以在真实设备开发完成后使用应用模拟器验证业务流，真实设备开发请参照[从这里开始](#)进行。

- 当设备侧开发和应用侧开发均未完成时，您可以创建虚拟设备，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：

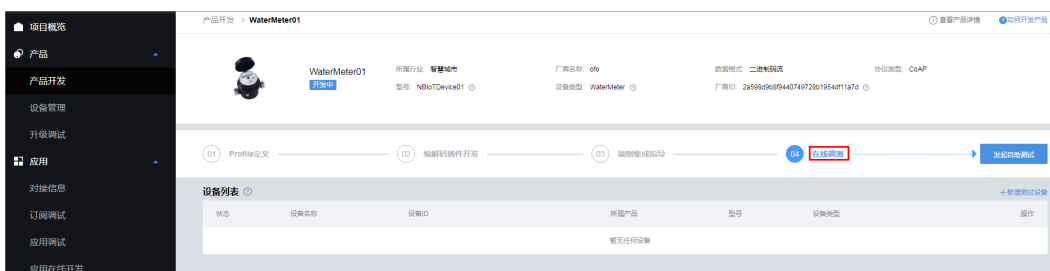


- 当设备侧开发已经完成时，但应用侧开发还未完成时，您可以创建真实设备，使用应用模拟器对设备、Profile、插件等进行调测。真实设备调测界面结构如下：



## 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”。



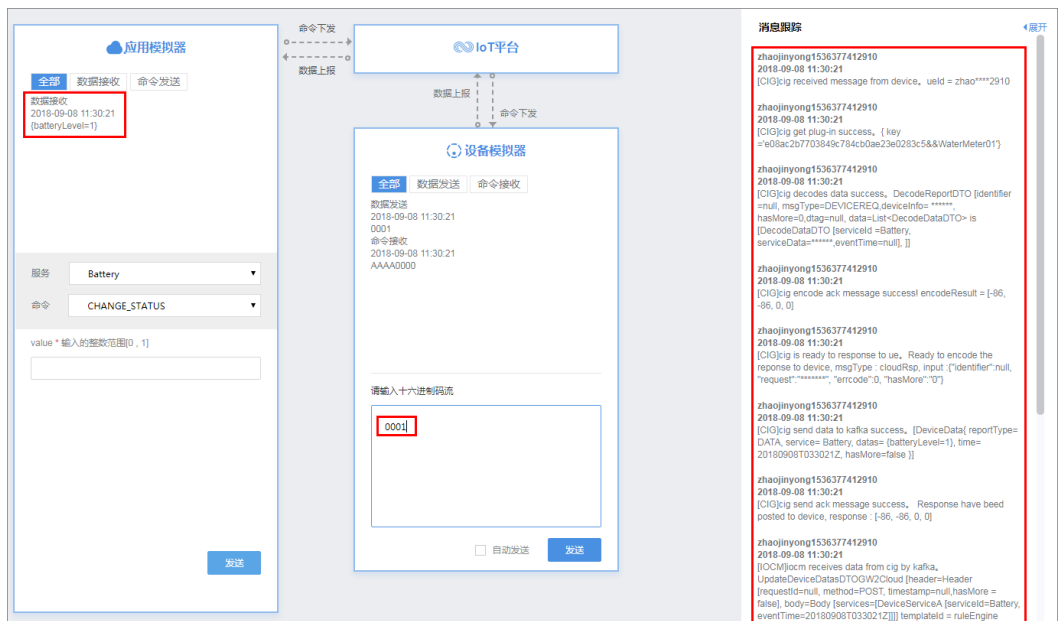
步骤3 系统将弹出“新增测试设备”窗口，勾选“没有真实的物理设备”，点击“创建”。



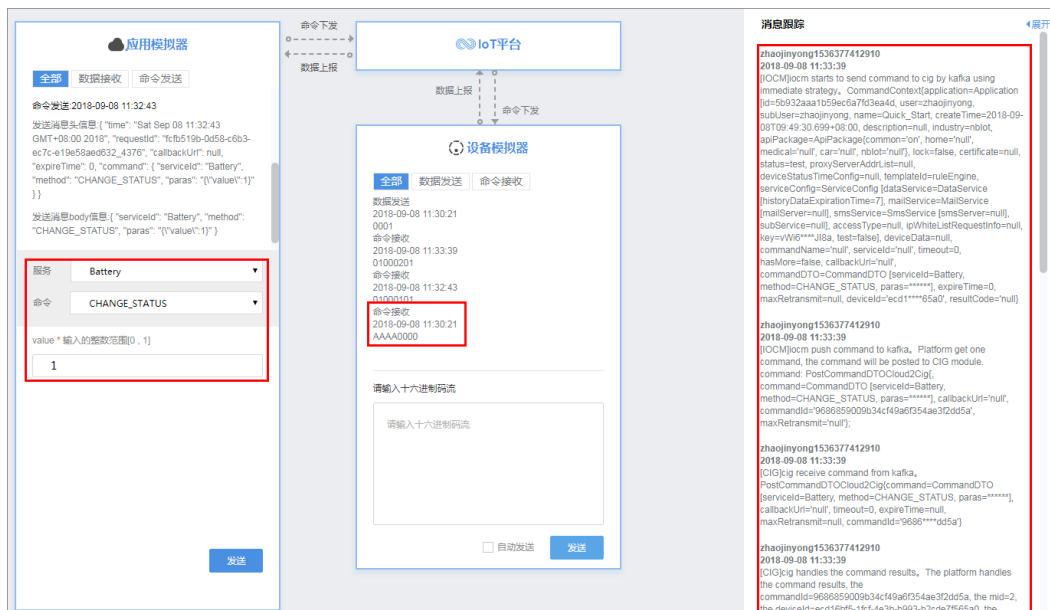
**步骤4** 在设备列表中，选择新创建的虚拟设备，进入调试界面。虚拟设备名称组成为：“产品名称”+“Simulator”，每款产品下只能创建一个虚拟设备。

状态	设备名称	设备ID	所属产品	产品型号	产品类型	操作
<input checked="" type="radio"/> 在线	WaterMeter01NBSimulator	d1ec9269-240c-4168-abe4-7bbaf66e583b	WaterMeter01	WaterMeter01	WaterMeter	
<input type="radio"/> 离线	testdevice001	c2c3fb2-8f5-48f1-b44e-5943ec64d575	WaterMeter01	WaterMeter01	WaterMeter	

**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。



**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。



----结束

## 使用真实设备调测

请先参照[设备侧开发](#)将设备接入开发中心，开发中心的接入地址请参照[平台对接信息](#)获取。

确保设备上报的数据信息与Profile中定义的一致，且如果设备上报的是二进制码流，请确保已经开发了编解码插件。

**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”。

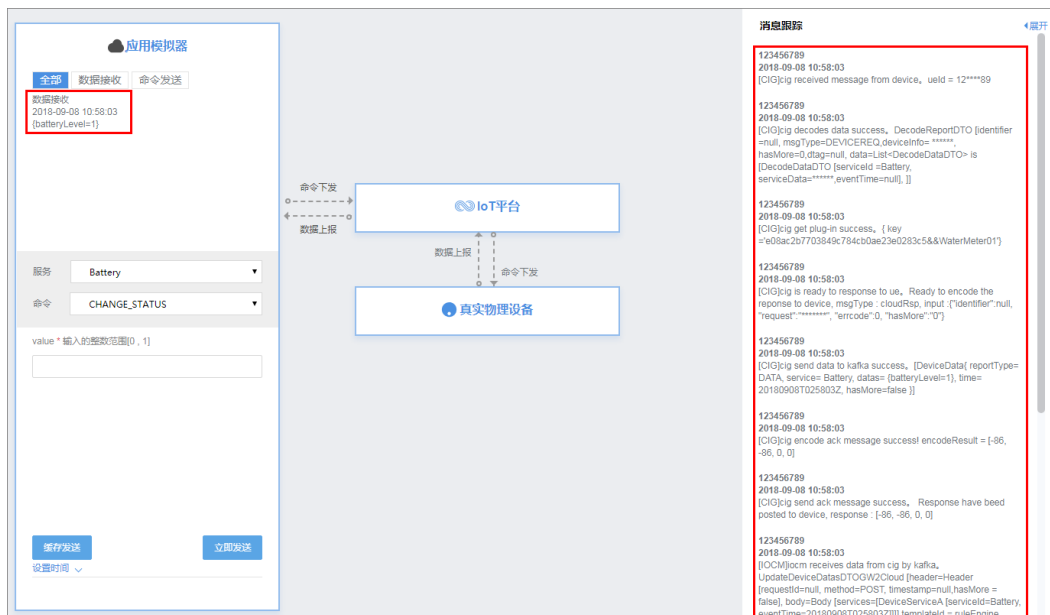
- “设备名称”只允许大小写字母、数字和下划线，需要在产品下保持唯一。
- “设备标识”需要在产品下保持唯一，如设备的IMEI、mac等。
- “验证码加密”根据设备的实际情况进行配置。

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。

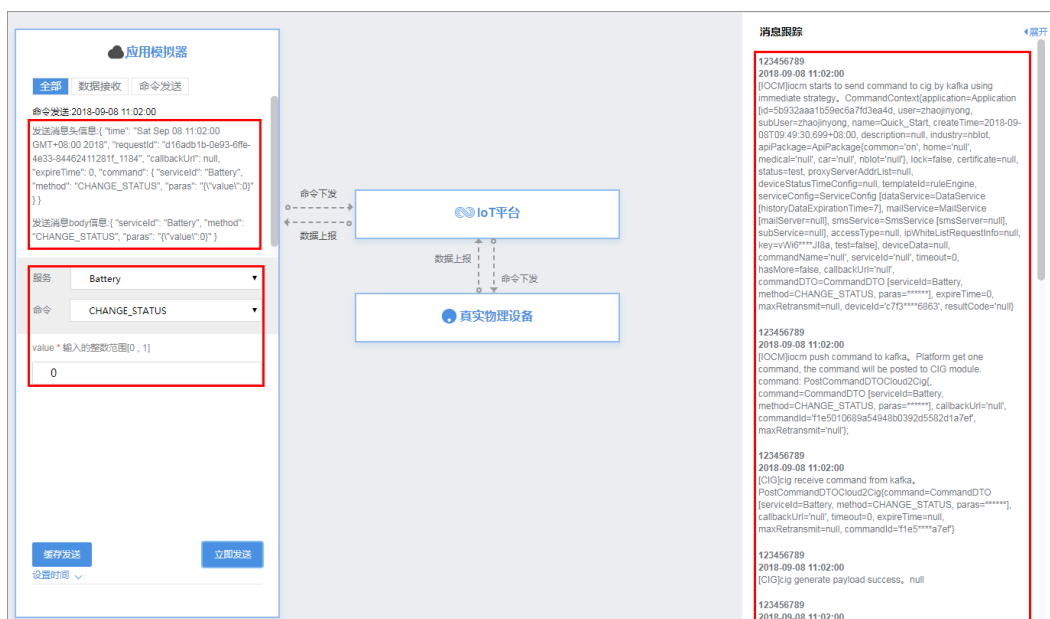
**步骤4** 在设备列表中，选择新创建的真实设备，进入调试界面。

状态	设备名称	设备ID	所属产品	产品型号	产品类型	操作
● 离线	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。



**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

## 设备管理

平台为每一个注册到平台上的设备生成了一个 **deviceId**，这是设备在平台上的唯一 ID，在接口调用时，每个与设备的相关操作都需要 **deviceId**。开发中心的“设备管理”功能模块呈现本项目中的所有产品下的真实设备和虚拟设备，并提供分类统计、在线调测、设备日志等功能，以便于进行设备管理和问题定位。

- 在“设备管理”新增真实设备，设备名称由开发者定义。当设备侧开发已经完成时，您可以在开发中心创建真实设备，对真实物理设备、编解码插件、应用服务器等进行端到端调试。

- 在“设备管理”新增虚拟设备，设备名称由系统生成，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。当设备侧开发还未完成时，您可以在开发中心创建虚拟设备，对编解码插件、应用服务器等进行调测。

## 4.6.1 设备侧开发

### 4.6.1.1 使用 MQTTS 协议接入（联通用户专用）

非联通用户请查看[设备接入服务](#)。

MQTTS是安全的基于TLS的加密协议，采用MQTTS协议接入平台的设备，设备与物联网平台之间的通信过程，数据都是加密的，具有一定的安全性。

MQTT主要应用于计算能力有限，且工作在低带宽、不可靠的网络的远程传感器和控制设备，适合长连接的场景，如智能路灯等。更多关于MQTT协议语法及接口信息，请访问<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/>获取。

#### 📖 说明

若选择MQTTS协议接入平台，建议通过[使用Agent SDK接入](#)。

## 使用限制

描述	限制
支持的MQTT协议版本	3.1.1
与标准MQTT协议的区别	<ul style="list-style-type: none"><li>支持Qos 0和Qos 1</li><li>不支持QoS2</li><li>不支持will、retain msg</li><li>不支持Topic自定义</li></ul>
MQTTS支持的安全等级	采用TCP通道基础 + TLS协议（TLSV1、TLSV1.1和TLSV1.2 版本）
单帐号每秒最大MQTT连接请求数	无限制
单个设备每分钟支持的最大MQTT连接数	1
单个MQTT连接每秒的吞吐量，即带宽，包含直连设备和网关	3KB/s
MQTT单个发布消息最大长度，超过此大小的发布请求将被直接拒绝	1MB
MQTT连接心跳时间建议值	心跳时间限定为30至1200秒，推荐设置为120秒
产品是否支持自定义Topic	不支持
消息发布与订阅	设备只能对自己的Topic进行消息发布与订阅



描述	限制
每个订阅请求的最大订阅数	无限制

## 相关资源

您可以参考[MQTT接口](#)文档，把具备MQTT通信能力的设备接入物联网平台。您也可以[通过MQTT.fx体验设备接入](#)，快速验证是否可以与物联网平台服务交互发布或订阅消息。

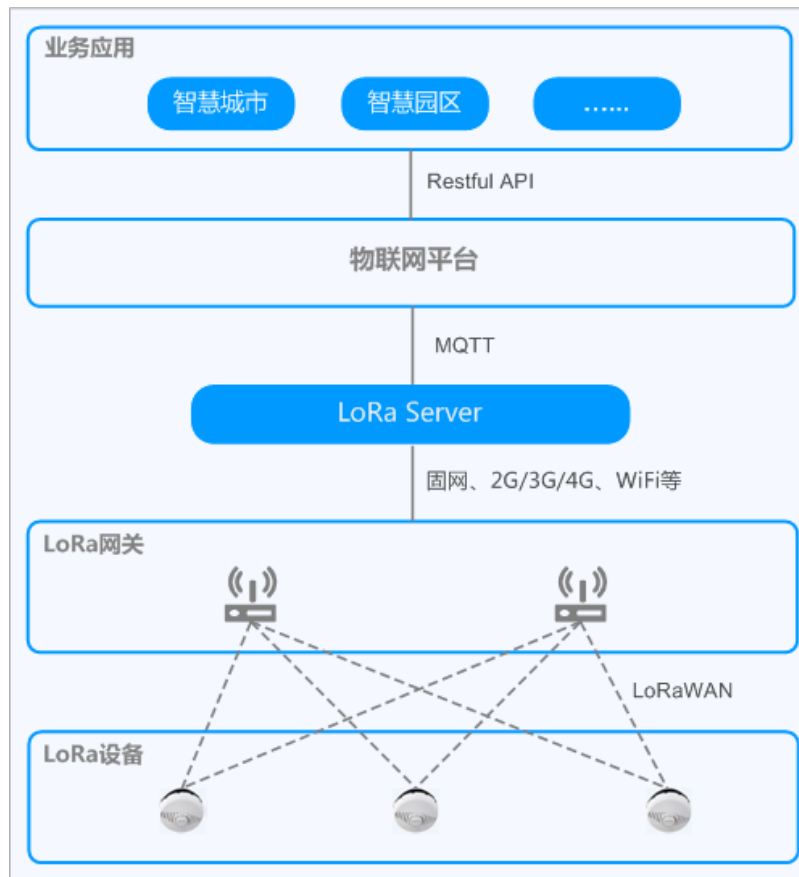
### 4.6.1.2 使用 LoRaWAN 协议接入（联通用户专用）

非联通用户请查看[设备接入服务](#)。

## 概述

LoRa（Long Range Radio）是当前应用较广的一种物联网无线接入技术，它最大的特点就是在同样功耗条件下比其他无线接入方式传播的距离更远，实现了低功耗和远距离的统一，它在同样的功耗下比传统的无线射频通信距离扩大3-5倍。而LoRaWAN是为LoRa远距离通信网络设计的一套通讯协议和系统架构。

物联网平台支持设备通过LoRa网络接入，但LoRaWAN设备不能直连物联网平台，需要经过LoRa服务商提供的LoRa Server接入物联网平台，如下图所示。



- LoRa Server: 由LoRa服务商提供的LoRa接入服务器, LoRa Server与物联网平台之间通过MQTT协议对接, 与LoRa网关之间通过标准IP网络对接。
- LoRa网关: LoRa网关向设备提供LoRa网络接入, 设备通过LoRa网关对接到LoRa Server。LoRa网关通常也由LoRa服务商提供。
- LoRaWAN设备: LoRaWAN设备即LoRa节点, 其通过LoRa网络接入LoRa网关, 然后通过LoRa Server最终接入到物联网平台。LoRaWAN设备可以通过以下几种方式获取:
  - 直接购买合适的LoRaWAN设备。
  - 购买LoRa模组, 并集成到设备中。

#### 📖 说明

LoRaWAN设备和LoRa网关并无一一对应关系, LoRaWAN设备可通过LoRa网络向任何一个LoRa网关发送数据。若多个LoRa网关同时接收到一个LoRaWAN设备的数据并上报到LoRa Server, LoRa Server会进行数据去重。

## 接入流程

#### 📖 说明

测试场景: 平台提供LoRaWAN协议的测试环境, 多个用户共用一个Network Server, 您可以通过提工单的方式获取测试账号, 目前账号使用期限为7天, 到期回收。

商用场景: LoRa服务商在华为云为每个客户部署一个Network Server。

#### 前置条件:

在LoRaWAN设备接入物联网平台前, 您需要完成以下前置条件:

- 选择LoRa服务商: 选择一个LoRa服务商并与其达成合作关系, 您可以通过[提交工单](#)咨询华为云物联网平台当前支持的LoRa服务商。
- 完成LoRa网络的对接调试: 在选择LoRa服务商后, 您需要根据LoRa服务商的要求和指导, 完成LoRaWAN设备、LoRa网关和LoRa Server的对接调试。

#### 接入流程:


步骤	说明
对接LoRa Server	LoRa Server作为一个MQTT设备在物联网平台上注册并上线。
创建LoRa网关	LoRa网关作为LoRa Server下的一个子设备, 在物联网平台上注册。
创建LoRaWAN设备	LoRaWAN设备作为LoRa Server下的一个子设备, 在物联网平台上注册。
业务调试	对接入物联网平台的LoRaWAN设备进行数据上报和命令下发测试, 具体操作可参考 <a href="#">数据上报</a> 和 <a href="#">命令下发</a> 。

## 对接 LoRa Server

在LoRaWAN设备接入物联网平台前, 需要先完成物联网平台与LoRa Server的对接。

**步骤1** 新增LoRa Server的产品模型。

您可在产品中心搜索“设备类型”为“loraServer”，“产品范围”为“第三方公开”的产品模型，导入使用；也可以按照以下步骤手动创建。

1. 登录[物联网平台控制台](#)，点击右上角“进入设备管理服务”。
2. 单击左下角，选择“产品模型”，单击“新增产品模型”，选择“手动创建”。
3. 在弹框页面里填写关键参数信息后，单击“确定”。
  - 产品名称：自定义该新建产品的产品名。
  - 型号：LoRa Server的型号，可自定义设置。
  - 设备类型：固定设置为“loraServer”。
  - 厂商名称：LoRa服务商的名称，可自定义设置。
  - 协议类型：选择“MQTT”。

**步骤2** 注册LoRa Server。

1. 在设备管理服务中选择“设备 > 设备注册”。
2. 选择页签“单个注册”，单击右上角“创建”，填写关键参数后，单击“确定”。
  - 选择产品：选择1中添加的LoRa Server产品模型。
  - 设备标识码：对于LoRa Server，该参数实际未使用，可自定义设置。
  - 密钥：可自定义设置。
  - 确认密钥：可自定义设置。
3. 注册设备成功后，将平台返回的“设备ID”和“密钥”保存并提供给LoRa服务商。

**步骤3** LoRa服务商部署LoRa Server，LoRa Server通过设备ID和密钥接入物联网平台。**步骤4** 在设备管理服务中选择“设备 > 所有设备”，确认上一步注册的LoRa Server设备状态为“在线”，则LoRa Server对接成功。


----结束

## 创建 LoRa 网关

在LoRa Server与物联网平台对接成功后，以子设备的方式在LoRa Server下创建LoRa网关。

在物联网平台上创建LoRa网关，平台会同步在LoRa Server上注册LoRa网关的信息，使LoRa网关可以正常接入LoRa Server。

**步骤1** 新增LoRa网关的产品模型。如果物联网平台上已添加了LoRa网关的产品模型，可跳过此步。

1. 登录[物联网平台控制台](#)，点击右上角“进入设备管理服务”。
2. 单击左下角，选择“产品模型”，单击“新增产品模型”，选择“手动创建”。
3. 在弹框页面里填写关键参数信息后，单击“确定”。

- 产品名称：自定义该新建产品的产品名。
- 型号：LoRa网关的型号，可自定义设置。
- 设备类型：固定配置为LoRaGateway。
- 厂商名称：LoRa网关的厂商名称，可自定义设置。
- 协议类型：选择“LoRaWAN”。

## 步骤2 添加LoRa网关。

1. 在设备管理服务中选择“设备 > 所有设备”，单击已注册的LoRa Server设备进入“设备详情”页。
2. 选择“子设备”页签，单击右上角的“添加”按钮添加子设备。
3. 在弹框页面里填写关键参数后，单击“确定”。
  - 选择产品：选择**步骤1**中添加的LoRa网关产品模型。
  - 设备名称：自定义该新建设备的名称。
  - 设备标识码 (gatewayEUI)：填写LoRa网关的gatewayEUI，从LoRa网关上获取。
  - 接入码：在LoRa Server上注册LoRa网关或LoRaWAN设备使用的接入码，由LoRa服务商提供。

### 说明

新注册的LoRa网关处于未激活状态。如果LoRa网关是MQTT协议，接入网络2分钟后，LoRa网关变为在线状态；如果LoRa网关是UDP协议，则一直处于未激活状态。


----结束

## 创建 LoRaWAN 设备

在LoRa Server与物联网平台对接成功和LoRa网关在线后，以子设备的方式在LoRa Server下创建LoRaWAN设备。

在物联网平台上创建LoRaWAN设备，平台会同步在LoRa Server上注册LoRaWAN设备的信息，使LoRaWAN设备可以正常接入LoRa Server。

**步骤1** 新增LoRaWAN设备的产品模型。如果物联网平台上已添加了LoRaWAN设备的产品模型，可跳过此步。

1. 登录**物联网平台控制台**，点击右上角“进入设备管理服务”。
2. 单击左下角 ，选择“产品模型”，单击“新增产品模型”，产品模型可通过手动创建和从本地导入两种方式。
  - 选择“本地导入”，进入到本地导入产品的页面。
    - i. 在弹出的窗口中输入产品名称，并上传Profile资源文件。
    - ii. 单击“确定”，等待导入完成。

### 说明

产品ID和产品密钥用于设备注册，请单击“保存密钥至本地”，以保存产品密钥信息，密钥信息在产品模型详情里不可见，请妥善保管。

- 选择“手动创建”，在弹框页面里填写关键参数信息后，单击“确定”。

- 产品名称：自定义该新建产品的产品名。
- 型号：LoRaWAN设备的型号，可自定义设置。
- 设备类型：非LoRaGateway的其他类型。
- 厂商名称：LoRaWAN设备的厂商名称，可自定义设置。
- 协议类型：选择“LoRaWAN”。

#### 步骤2 添加LoRaWAN设备。

1. 在设备管理服务中选择“设备 > 所有设备”，单击已注册的LoRa Server设备进入“设备详情”页。
2. 选择“子设备”页签，单击右上角的“添加”按钮添加子设备。
3. 在弹框页面里填写关键参数后，单击“确定”。
  - 选择产品：选择步骤1中添加的LoRaWAN设备产品模型。
  - 设备名称：自定义该新建设备的名称。
  - 设备标识码 (devEUI)：填写LoRaWAN设备的devEUI，在LoRaWAN设备上获取。
  - 接入码：在LoRa Server上注册LoRa网关或LoRaWAN设备使用的接入码，由LoRa服务商提供。
  - 设备工作模式：可选择classA，classB或classC，根据LoRaWAN设备的实际设置选择。
  - appEUI：LoRa应用ID，购买LoRaWAN设备时提供的产品参数单上获取。
  - 激活方式：可选择ABP或OTAA方式，根据LoRaWAN设备的实际设置选择。
  - appSKey：激活方式选择ABP时需要填写，购买LoRaWAN设备时提供的产品参数单上获取。
  - devAddr：激活方式选择ABP时需要填写，购买LoRaWAN设备时提供的产品参数单上获取。
  - nwkSKey：激活方式选择ABP时需要填写，购买LoRaWAN设备时提供的产品参数单上获取。
  - appKey：激活方式选择OTAA时需要填写，购买LoRaWAN设备时提供的产品参数单上获取。

#### 📖 说明

新注册的设备处于未激活状态，当上报数据后变为在线状态。

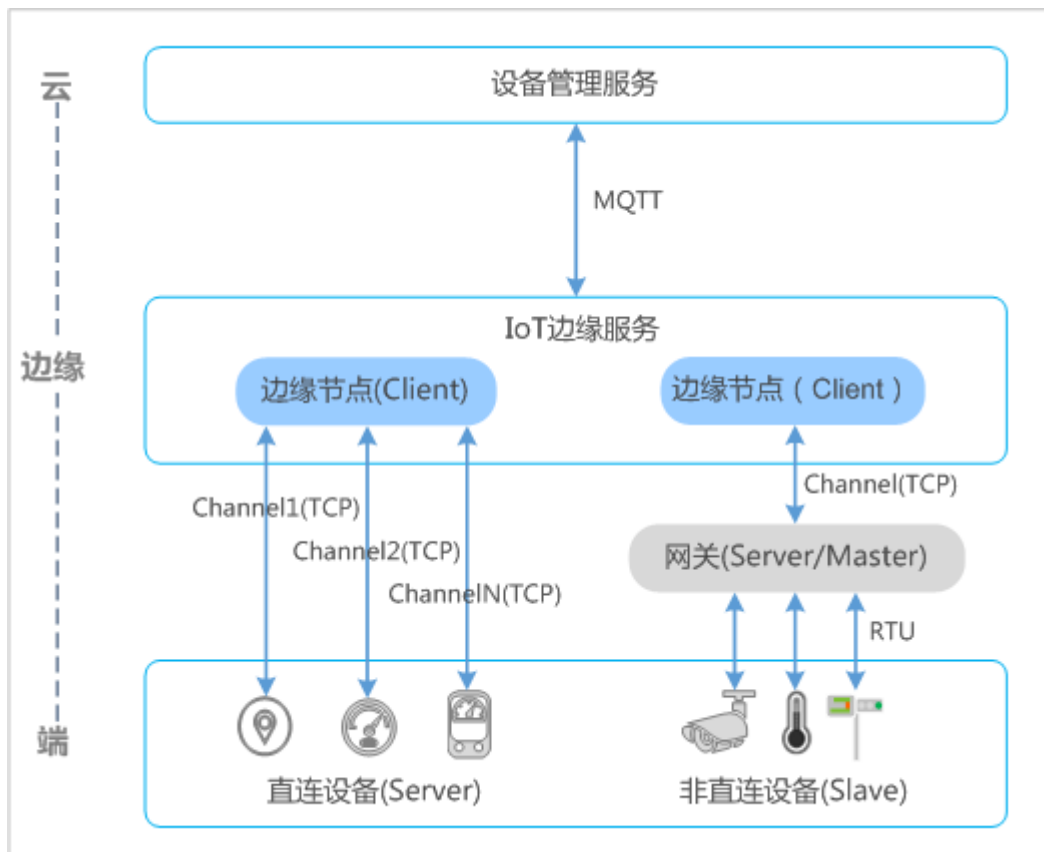
----结束

### 4.6.1.3 使用 Modbus 协议接入（联通用户专用）

非联通用户请查看[设备接入服务](#)。

#### 概述

Modbus协议目前已经成为工业领域通信协议的业界标准，是工业电子设备之间最常用的连接方式。使用Modbus协议的方式包括直连和非直连两种方式，组网方案如下图所示。



接入方式	方案说明	适用场景
直连接入	Modbus设备直接与边缘节点进行连接，通过TCP协议通信。边缘节点作为客户端，Modbus设备作为服务端进行连接。该方式中，在传输通道绑定设备时，边缘节点与Modbus设备之间的通道只能绑定一个设备。	Modbus设备具备TCP通信能力。 <b>说明</b> Modbus设备具备TCP能力，但如果IP资源比较受限，无法为每一个Modbus设备分配一个IP地址，也可采用非直连方式进行连接组网。
非直连接入	Modbus设备通过RTU串口与网关进行连接，然后边缘节点通过TCP协议与网关进行通信。网关作为Master节点，Modbus设备作为Slave节点进行连接。该方式中，在传输通道（边缘节点与网关之间的TCP传输通道）绑定设备时，需要将该网关下的Modbus设备绑定到指定的通道，并且，在同一个传输通道中，通过“从站号”区分不同的Modbus设备。	Modbus设备不具备TCP通信能力，只能通过RTU（串口）方式进行通信。

在数据采集时，边缘节点可根据指定的数据采集间隔时间，主动向Modbus设备或通过网关向Modbus设备进行数据的采集，数据格式为二进制格式。数据采集完成后，IoT边缘服务将采集的数据规整（Json格式）后，通过MQTT的方式将数据上报到设备管理。

## 接入流程

业务场景	操作步骤	说明
设备准备	准备Modbus网关	购买支持Modbus协议的网关（Modbus设备与网关支持通过RTU通信，网关与边缘节点采用TCP通信）
	准备Modbus设备	购买支持Modbus协议的传感器设备
物联网平台侧操作	开发Modbus设备产品模型	在产品中心开发Modbus设备的产品模型
	添加Modbus设备	在边缘节点下，添加Modbus设备
	创建Modbus通道和绑定设备	将Modbus设备绑定到指定的TCP传输通道

## 开发 Modbus 设备产品模型

产品模型（也称Profile）是用于描述Modbus设备具备的能力，通过定义产品模型，在物联网平台构建一款设备的抽象[在线开发Profile](#)型，使平台理解该款Modbus设备支持的服务、属性、命令等信息，如温度、电量等。Modbus设备的产品模型定义分为两部分：包含Modbus设备的能力（图中红色框内容）和定义Modbus点位表（图中蓝色框内容），如图所示。属性和命令的定义请根据[在线开发Profile](#)，在定义属性和命令的基础上，再定义Modbus点位表，下面将详细介绍点位表的定义方法。

### 新增属性



\* 名称

\* 数据类型

\* 最小值  \* 最大值

步长  单位

### 新增扩展描述

modbus读功能码  modbus写功能码

\* 寄存器地址

\* 寄存器数量

\* 交换寄存器内高低字节

\* 交换寄存器顺序

\* 缩放因子

\* 访问模式

- R 属性值可读
- W 属性值可写 (更改)
- E 属性值更改时上报事件

确定

取消



**步骤1** 登录[物联网平台控制台](#)，点击右上角进入“开发中心”。

**步骤2** 定义Modbus设备的属性和Modbus点位表，属性的定义请参考[在线开发Profile文件](#)，然后定义定位表（上图中的新增扩展描述）字段，原则如下表所示。

参数名称	参数说明
Modbus读功能码	<p>读功能码分为“比特访问（Bit Access）”和“16比特访问（16 Bit Access）”，功能码均为十进制。</p> <p>比特访问功能码：</p> <ul style="list-style-type: none"> <li>● 01：读线圈（Read Coils）</li> <li>● 02：读输入离散量（Read Discrete Inputs）</li> </ul> <p>16比特访问功能码：</p> <ul style="list-style-type: none"> <li>● 03：读多个寄存器（Read Holding Registers）</li> <li>● 04：读单个寄存器(Read Input Register)</li> </ul>
Modbus写功能码	<p>写功能码分为“比特访问（Bit Access）”和“16比特访问（16 Bit Access）”，功能码均为十进制。</p> <p><b>说明</b> 定义属性时，无需填写该字段。定义命令时，需要填写。</p> <p>比特访问功能码：</p> <ul style="list-style-type: none"> <li>● 05：写单个线圈（Write Single Coil）</li> <li>● 15：写多个线圈（Write Multiple Coils）</li> </ul> <p>16比特访问功能码：</p> <ul style="list-style-type: none"> <li>● 06：写单个寄存器（Write Single Register）</li> <li>● 16：写多个寄存器（Write Multiple Registers）</li> </ul>
寄存器起始地址	寄存器起始地址，占16比特，例如：00 01
寄存器数量	属性对应的数据所在的寄存器个数，占16比特，例如：00 02
交换寄存器内高低字节	是否对寄存器内的数据进行高低位交换，默认：True。例如寄存器中存储的属性数据为：10011011，则IoT边缘节点获取的数据经过高低位交互后为：01100111。
交互寄存器顺序	是否对寄存器的位置进行交换，默认：True。例如寄存器的起始地址为0001，寄存器数量为0002，则将0002地址寄存器与0001地址寄存器进行交换。
缩放因子	对寄存器内的数据进行乘以缩放因子，得到所需的数据，例如获取的温度数据为365，缩放因子为0.1，则得到实际的温度数据为 $365 \times 0.1 = 36.5$

**步骤3** 定义Modbus设备的属性和Modbus点位表，命令的定义请参考[在线开发Profile文件](#)，然后定义定位表（上图中的新增扩展描述）字段，点位表的填写原则如**步骤2**中的表格所示。

**步骤4** 定义完属性和命令后，请将Profile文件进行导入，然后根据[添加Modbus设备](#)操作，在设备管理服务中导入开发的Modbus设备产品模型。



----结束

## 添加 Modbus 设备


前提条件:

- 已完成IoT边缘应用的部署。
- 已在开发中心完成Modbus设备产品模型开发。

操作步骤:

部署完IoT边缘应用后，在设备管理服务控制台的设备列表中，会自动生成边缘节点设备，现在需要在边缘节点设备下，添加Modbus设备。

**步骤1** 登录[物联网平台控制台](#)，点击右上角“进入设备管理服务”。

**步骤2** 单击左下角，在“产品模型”界面中，单击右上角的“新增产品模型”，选择“本地导入”，将Modbus设备的产品模型导入到设备管理中。

**步骤3** 选择IoT边缘节点所属的行业应用，并在“设备 > 所有设备”中，找到边缘节点设备。

**步骤4** 单击边缘节点设备，进入设备详情界面，在“子设备”页签中，单击“添加”按钮，创建Modbus设备。

**步骤5** 在弹出的对话框中，选择上传的Modbus设备的产品模型，填写设备名称后，单击“确定”，完成Modbus设备的添加。添加完成后，Modbus设备的状态为“离线”。

### 说明

Modbus设备的状态由IoT边缘节点进行上报，如果IoT边缘节点不能正常上报子设备的状态信息到设备管理服务，则展示的子设备状态不会刷新。

- 在线：设备管理服务采集到Modbus设备上报的数据后，状态就会刷新为在线。
- 离线：如果IoT边缘节点连续5个[数据采集间隔](#)无法获取Modbus设备的数据，则IoT边缘节点上报Modbus设备的状态为离线，设备管理服务将设备的状态刷新为离线。

----结束

## 为 Modbus 设备绑定通道

根据[概述](#)中的Modbus接入组网方案可以知道，Modbus设备在接入IoT边缘节点时，需要将Modbus设备绑定到具体的通道中。如果Modbus设备与IoT边缘节点直连，则该通道只能绑定该设备；如果Modbus设备与IoT边缘节点通过网关连接，则需要创建IoT边缘节点与网关之间的传输通道，并将Modbus设备绑定到指定的通道中。

**步骤1** 在边缘节点设备的详情界面中，选择“配置”页签。

**步骤2** 单击右上角的“通道”按钮，在弹出的对话框中单击“添加”按钮，填写通道信息后单击“确定”，完成TCP传输通道的添加。

参数名称	参数说明
通道名称	定义IoT边缘节点与Modbus设备或网关之间的传输通道。如：Channel_TCP01
传输模式	固定选择“TCP”。
IP地址	IoT边缘节点与Modbus设备或网关建立TCP传输通道时，IoT边缘节点作为客户端，Modbus设备或网关作为服务端，因此该地址需要填写为Modbus设备（直连）或网关（Modbus设备非直连接入IoT边缘节点）的IP地址。如：192.168.10.11
端口	TCP服务端（Modbus设备或网关）的端口号。如：8000

#### 📖 说明

Modbus设备通过网关接入到IoT边缘节点时，如果添加设备前，已经创建了相应的通道，可以不用再执行该步骤，直接按照**步骤3**绑定Modbus设备即可。

**步骤3** 在“配置”页签中，单击“绑定子设备”按钮，然后单击“添加”按钮。

**步骤4** 在弹出的添加子设备配置页面中，选择需要绑定的Modbus设备，然后点击“下一步”。

**步骤5** 在弹出的对话框中，填写相关信息后，单击“提交”，完成Modbus设备的通道绑定。配置完成后，设备管理服务会将配置信息下发给IoT边缘节点。

参数名称	参数说明
关联通道	选择需要将Modbus设备绑定的通道，如：Channel_TCP01
从站号	用于标识同一个通道下的不同Modbus设备。填写原则：需要与Modbus设备规划的从站号保持一致。
数据采集间隔	IoT边缘节点采集Modbus设备数据的时间间隔，单位为：秒，最小采集间隔可设置为1秒。请根据Modbus设备数据采集的实际周期进行灵活设置。

#### ---结束

执行完以上操作后，Modbus设备上电接入到IoT边缘节点后，经过一个数据采集周期即可在设备管理服务的设备列表中查看采集的设备数据。

## 4.6.1.4 使用 Agent SDK 接入

### 4.6.1.4.1 Agent Lite SDK 使用指南（C）（联通用户专用）

非联通用户请查看[设备接入服务](#)。

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。

## 使用必读

### 开发环境要求：

操作系统	工具链
当前支持的系统： <ul style="list-style-type: none"> <li>• ARM Linux (Embedded Linux)</li> <li>• MIPS Linux (Embedded Linux)</li> <li>• x86 Linux</li> <li>• x86_64 Linux</li> <li>• x86 Windows</li> <li>• x86_64 Windows</li> </ul>	需要支持以下工具链之一： <ul style="list-style-type: none"> <li>• gcc-linaro-arm-linux-gnueabi/raspbian</li> <li>• arm-none-linux-gnueabi</li> <li>• arm-linux-uclibceabi</li> </ul>

### Demo工程目录结构及文件说明：

目录结构	目录	说明
Demo	demo.c	demo源文件
	demo.h	demo头文件
├─demo.c	*.h	Agent Lite头文件
├─demo.h	libcrypto.so/libssl.so	openssl库文件
├─*.h	libuspsdk.so	Agent Lite编译后的库文件
├─libcrypto.so/libssl.so	conf	存放TLS证书文件
├─libuspsdk.so		
└─conf		

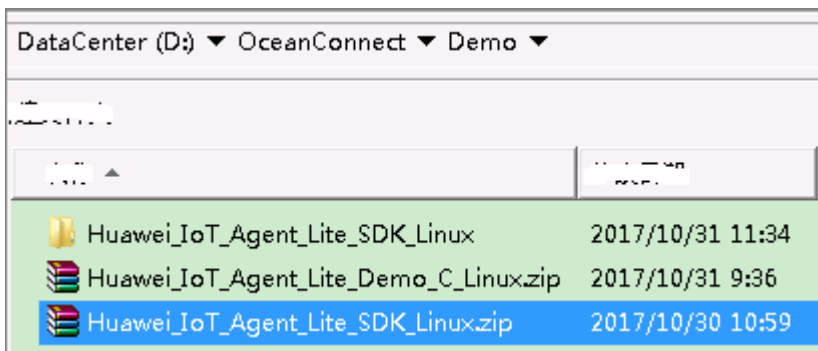
### 📖 说明

如果开发者没有设备，可以直接在X86 Linux系统进行开发。

### 交叉编译环境检测：

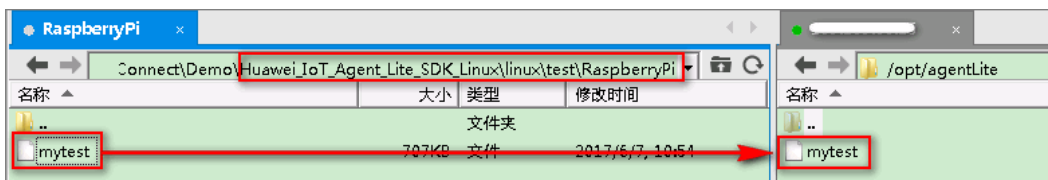
**步骤1** 准备网关或设备。本文档将以树莓派为例，说明如何集成网关。

**步骤2** 将Agent Lite SDK (Linux)解压到本地。



**步骤3** 用sftp工具把mytest测试工具上传到树莓派上。

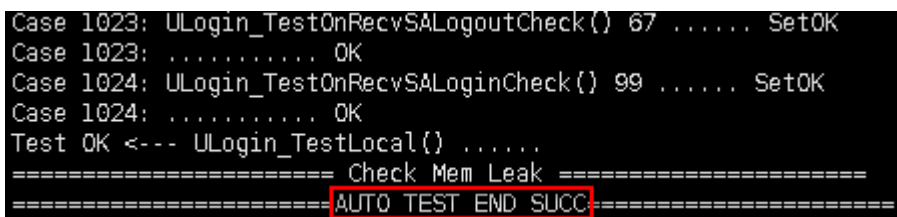
因为使用的设备是树莓派，所以使用RaspberryPi目录下的mytest测试工具。开发者可以选择根据实际情况选择不同目录下的mytest测试工具进行测试，尽量选择与自己的系统信息相近的目录下的测试工具进行测试。



**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。

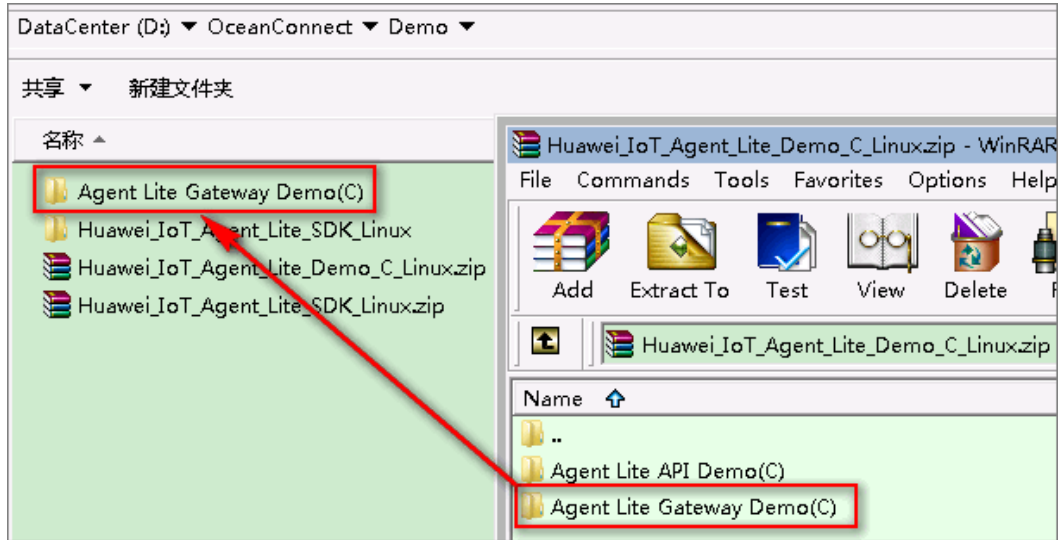


----结束

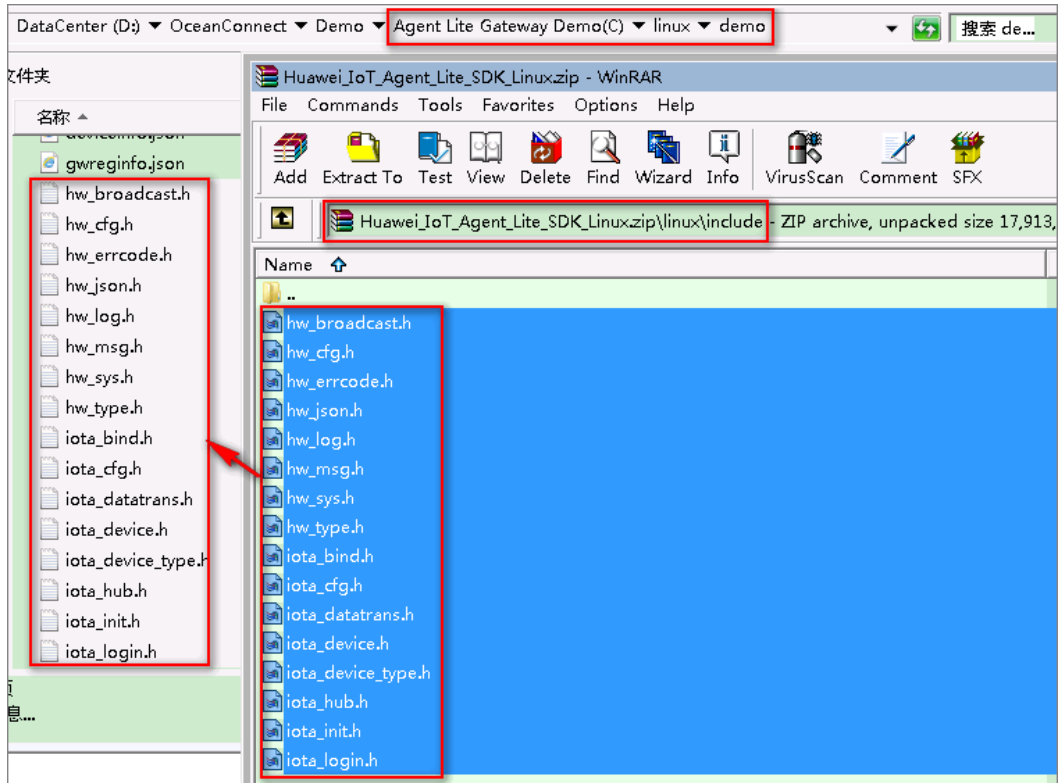
## 导入样例代码

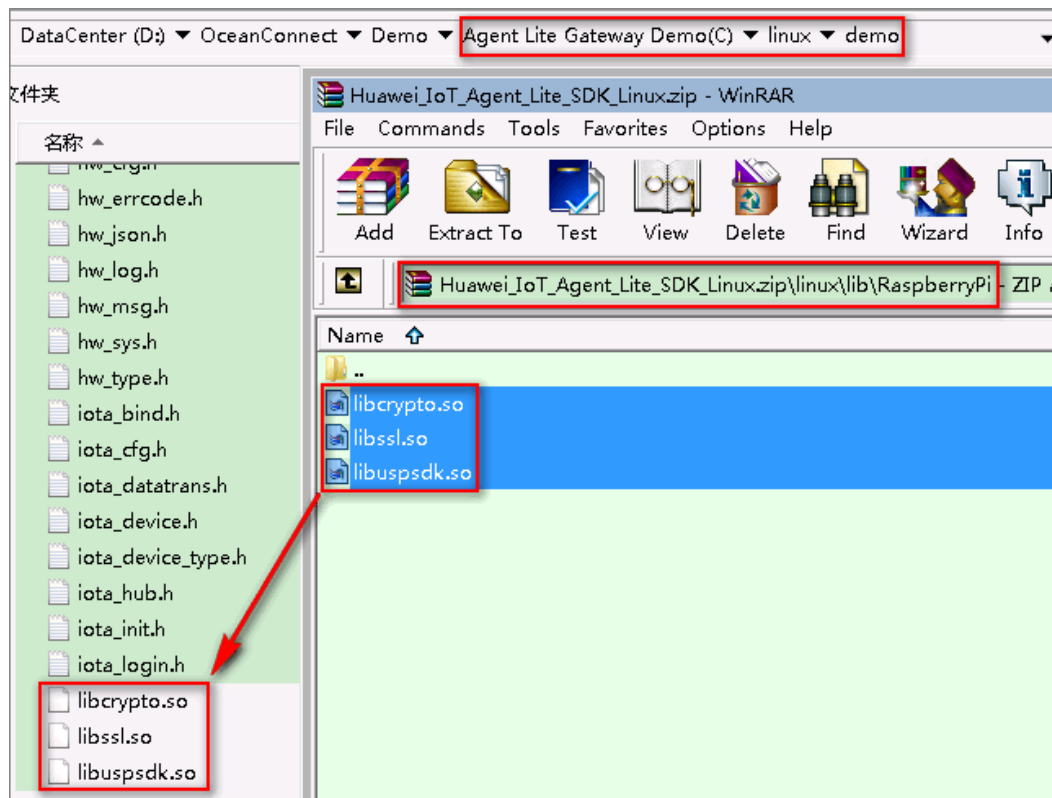
在交叉编译环境检测通过后，再进行设备集成开发。下面将以Agent Lite Gateway Demo(C)中的样例代码为例，说明如何进行Agent Lite SDK的集成。

**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。

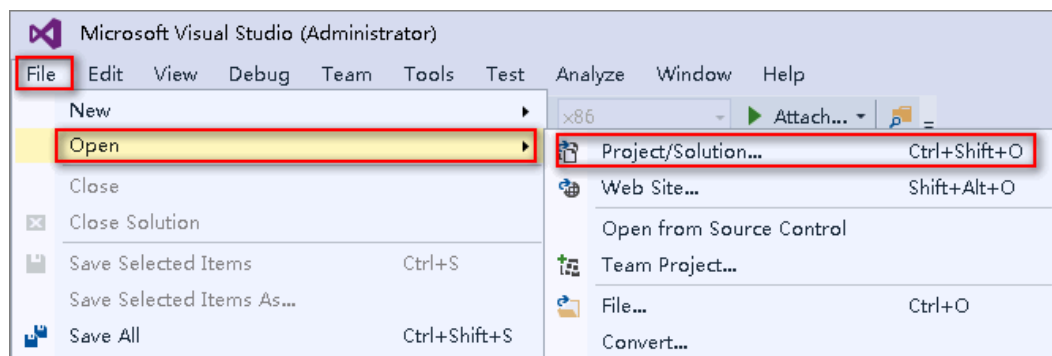




**说明**

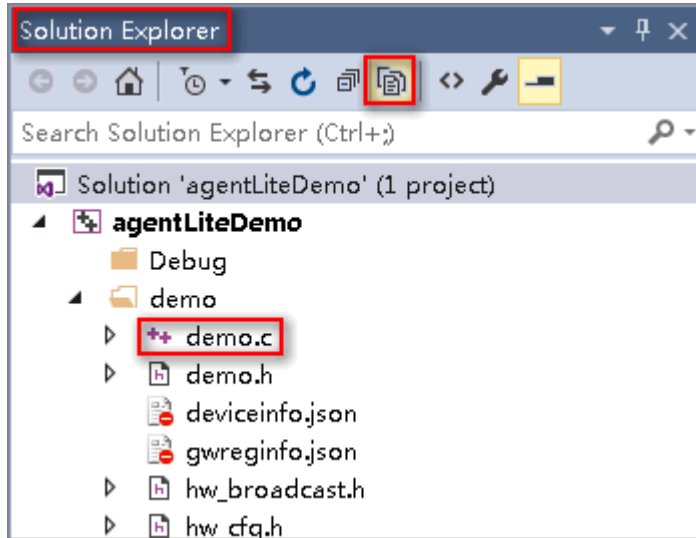
因为使用的设备是树莓派，所以使用RaspberryPi目录下的库。

**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

## 初始化

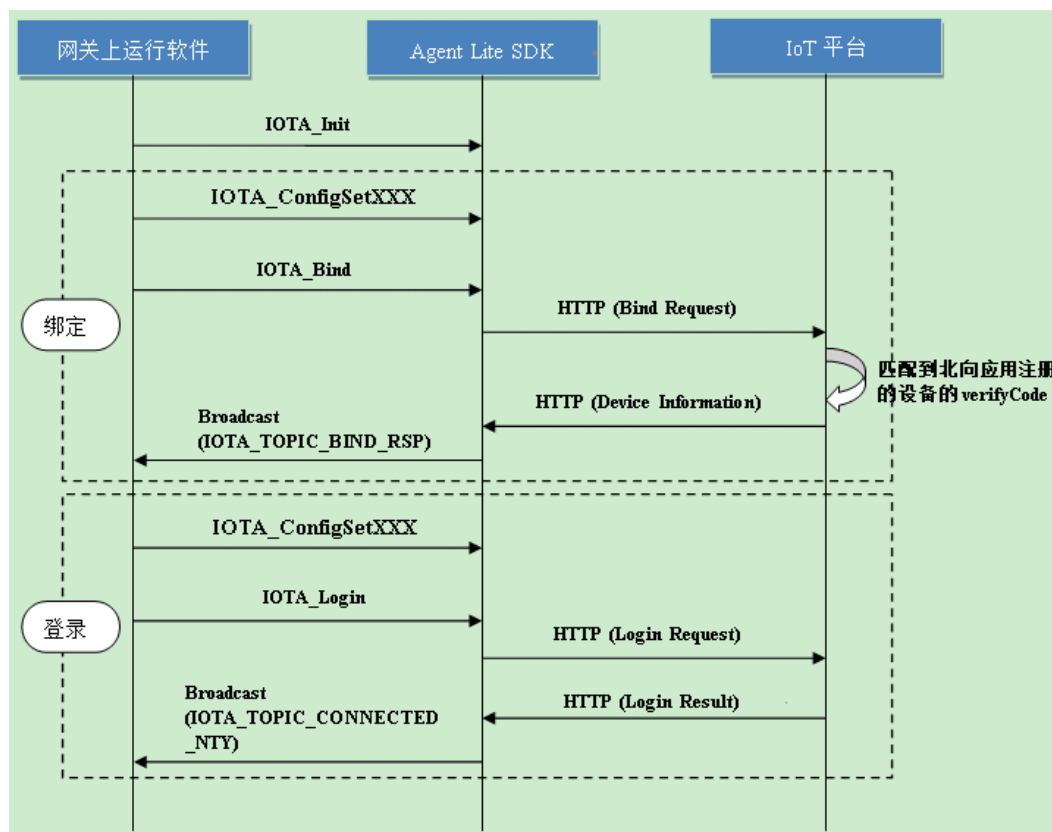
在发起业务前，需要先初始化Agent Lite相关资源，调用API接口IOTA\_Init()，初始化Agent Lite资源。具体API接口的参数使用请参考Agent Lite API接口文档。可参考demo.c中main()方法对IOTA\_Init()的调用。

```
IOTA_Init(const CONFIG_PATH,const HW_NULL);
```

- “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’ 。



## 绑定和登录



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。

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。

### 步骤1 修改绑定参数。

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“gwreginfo.json”文件中的如下信息：

- “platformaddr”：物联网平台的设备对接地址（MQTTS），可参考[平台对接信息](#)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。

#### 📖 说明

- 如果开发者通过“设备管理服务控制台”注册设备，则“mac”填写为设备注册时的“preSecret”（预置密钥）。
- 如果通过开发中心注册设备，则“verifyCode”填写为设备注册时设置的“nodeId”（设备标识）。
- “manufacturerId”（厂商Id）、“deviceType”（设备类型）、“model”（设备模型）和“protocolType”（协议类型）与Profile文件中的定义保持一致。

“gwreginfo.json”文件中设备固有示例：

```
{
  "mac": "1234567",
  "platformAddr": "127.0.0.1",
  "platformPort": 8943,
  "manufacturerId": "Huawei",
  "deviceType": "Gateway",
  "model": "AgentLite01",
  "protocolType": "HuaweiM2M",
  "loglevel": 255
}
```

## 步骤2 绑定设备。

绑定前先调用API接口**IOTA\_ConfigSetXXX()**设置物联网平台的IP与端口。

```
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API接口**IOTA\_Bind()**进行设备绑定，主要入参为MAC地址和必要的设备信息，包括“nodeId”（设备标识码）、“manufacturerId”（厂商Id）、“deviceType”（设备类型）、“model”（设备模型）和“protocolType”（协议类型），其中MAC地址与“nodeId”的值保持一致。

```
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);
}
```

设备或网关绑定成功，后续就不需要再绑定了，除非设备或网关被删除，才需要重新绑定。

设备绑定成功会收到广播，广播内容请参考Agent Lite API接口文档中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“gwbindinfo.json”文件中。

## 步骤3 配置登录参数。

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。

- “设备Id”（“EN\_IOTA\_CFG\_DEVICEID”）， “appld”（“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。

```
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);
```

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);
```

#### 步骤4 设备登录。

调用API接口**LoginService.login()**进行直连设备登录，具体API的参数使用参考Agent Lite API接口文档中设备登录接口的说明。

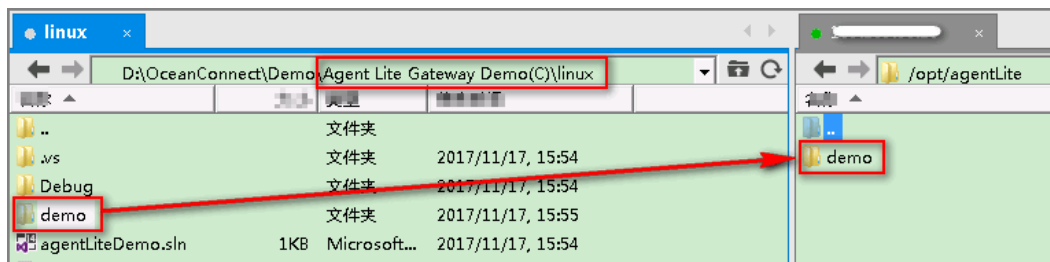
```
IOTA_Login();
```

----结束

## 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sftp工具把demo目录上传到树莓派上。



#### 步骤2 进行交叉编译。

1. 进入“/opt/agentLite/demo/”路径。  
“/opt/agentLite/demo/”路径需要根据实际情况修改。
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 -luspsdk -lssl -lcrypto -o sdk.out”命令，如果交叉编译成功，会产生sdk.out库文件。  
“luspsdk”、“lssl”和“lcrypto”分别表示库文件“libuspsdk.so”、“libssl.so”和“libcrypto.so”，具体linux交叉编译命令gcc请开发者自行学习。

```
root@raspberrypi:~# cd /opt/agentLite/demo/
root@raspberrypi:~# cd /opt/agentLite/demo $chmod -R 777 *
root@raspberrypi:~# cd /opt/agentLite/demo $export LD_LIBRARY_PATH=./
root@raspberrypi:~# cd /opt/agentLite/demo $gcc demo.c -L /opt/agentLite/demo/ -lpthread -ldl -lrt -luspsdk -lssl -lcrypto -o sdk.out
root@raspberrypi:~# cd /opt/agentLite/demo $
```

#### 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。

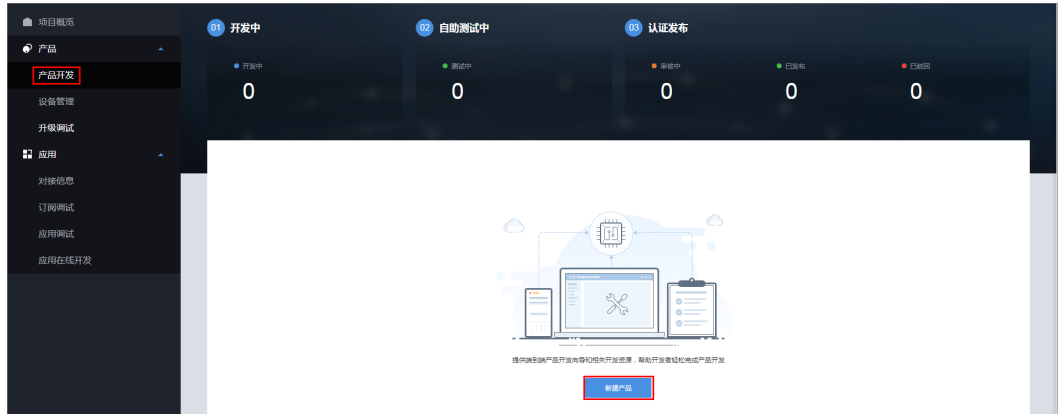
程序运行后，Agent Lite SDK就会主动发送绑定消息给平台。因为此时还没有使用应用侧接口在平台上注册此设备，所以会提示“The device has not registered yet”。

----结束

## 上传 Profile 并注册设备

下载**Profile开发示例**，并上传模板中的profile文件：  
“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

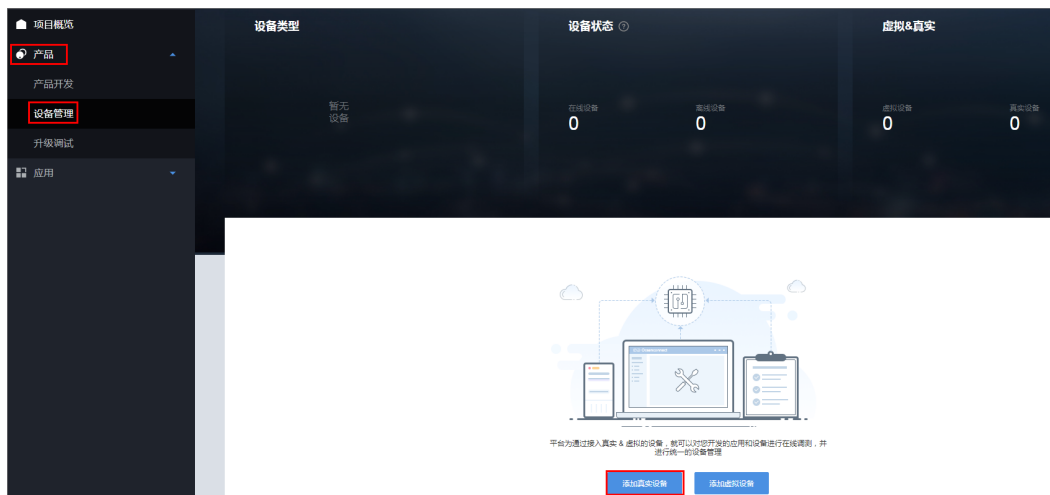
**步骤1** 登录开发中心，创建一个项目，在该项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“创建产品”中，选择“本地导入产品创建”，单击“上传Profile”上传“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。



**步骤3** 选择“产品 > 设备管理”，单击“添加真实设备”，进入“新建增实设备”页面。



**步骤4** 根据向导注册设备。

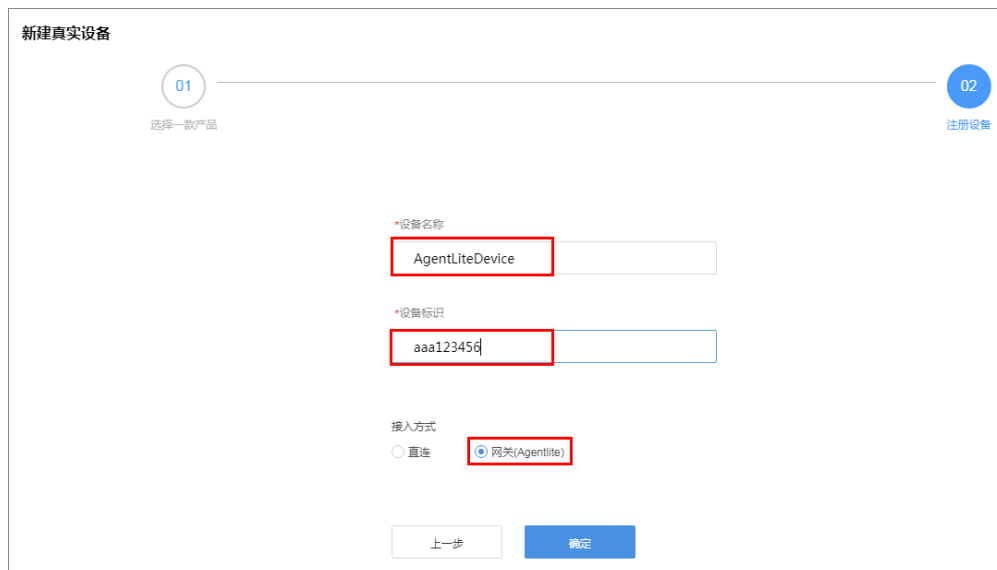
1. 选择产品。

产品：AgentLite001



2. 填写设备相关信息，单击“确定”。

- 设备名称：AgentLiteDevice
- 设备标识：aaa123456，需要与AgentLiteDemo中网关的设备标识一致。
- 接入方式：网关（Agentlite）



**步骤5** 注册设备后，Agent Lite SDK发送bind消息，则在开发中心的“产品 > 设备管理 > 设备列表”界面可以看到设备状态变成“在线”。



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。

----结束

## 数据上报和数据发布

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：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。

设备或网关登录成功后可以调用IOTA\_ServiceDataReport接口上报数据。

- 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。

```
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\":20240,\"usedPercent\":20}");
}
HW_INT Device_ServiceDataReport(const HW_CHAR *pcSensorDeviceID, const HW_CHAR
*pcServiceId, const HW_CHAR *pcServiceProperties)
{
    HW_CHAR aszRequestId[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(), aszRequestId, 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。广播过滤参数为**  
“IOTA\_TOPIC\_DATATRANS\_REPORT\_RSP/{deviceId}”。

```
sprintf(acBuf, "%s/%s", IOTA_TOPIC_DATATRANS_REPORT_RSP, g_stGateWayInfo.pcDeviceID);
HW_BroadCastReg(acBuf, Device_ServiceDataReportResultHandler);
```

- “数据发布”接口：topic固定为“/cloud/signaltrans/v2/categories/data”；“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。

设备或网关登录成功后可以调用**IOTA\_MqttDataPub**接口发布数据。

- “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。

```
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。广播过滤参数为**  
“IOTA\_TOPIC\_DATATRANS\_PUB\_RSP”。

```
HW_BroadCastReg(IOTA_TOPIC_DATATRANS_PUB_RSP, Device_ServiceDataPubResultHandler);
```

## 命令接收

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

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;  
}
```

在开发中心的“产品 > 设备管理”界面中，单击非直连设备列后的“删除”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。

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，所以网关收到删除非直连设备的命令也不会立刻将设备删除。

## 添加非直连设备

在添加非直连设备前，确认非直连设备的profile已经上传，详见[上传Profile并注册设备](#)步骤。

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。

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。

```
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。添加非直连设备成功后就能从广播中得到非直连设备的deviceld。

```
HW_BroadCastReg(IOTA_TOPIC_HUB_ADDDEV_RSP, Device_AddResultHandler);
```

### 📖 说明

Demo在设备添加成功一段时间后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。

非直连设备添加成功后可以在“设备列表”中看到新增一条记录。



## 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_cDeviceld, "ONLINE", "NONE");
```

AgentLiteDemo中只添加了一个非直连设备，所以IOTA\_DeviceStatusUpdate中的g\_cDeviceld直接使用全局变量。

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);
```

## 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“历史数据”中查看上报的数据。



### 4.6.1.4.2 Agent Lite SDK 使用指南（Java）（联通用户专用）

非联通用户请查看[设备接入服务](#)。

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。

## 使用必读

**开发环境要求：**

使用的SDK版本为jdk1.8.0\_45，适用的操作系统为Windows系统。

**工程目录结构及文件说明：**

目录结构	目录	说明
AgentLiteDemo   └──src     ├──com         ├──huawei         ├──agentlitedemo     └──libs         ├──agentlite-0.0.1-         SNAPSHOT         ├──usp_agentlite.jar         ├──.dll └──workdir	src	存放Agentlite Demo代码。
	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）。
	workdir	存放工程日志文件，由agentlite初始化资源设置。

目录结构	目录	说明
└─ 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。

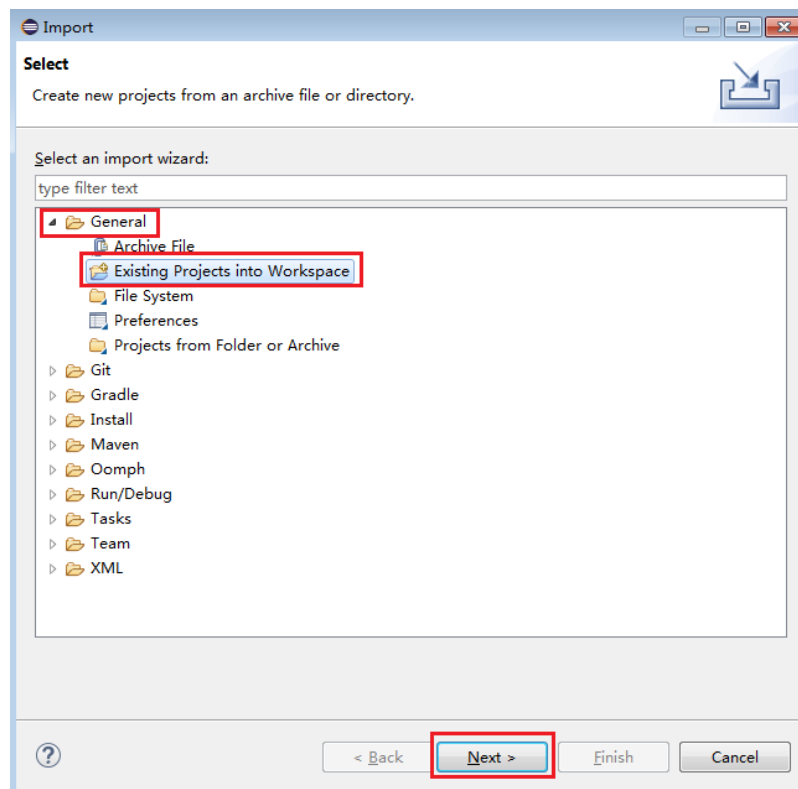
## 导入代码样例

**步骤1** 将Agent Lite Demo(Java)解压到本地。

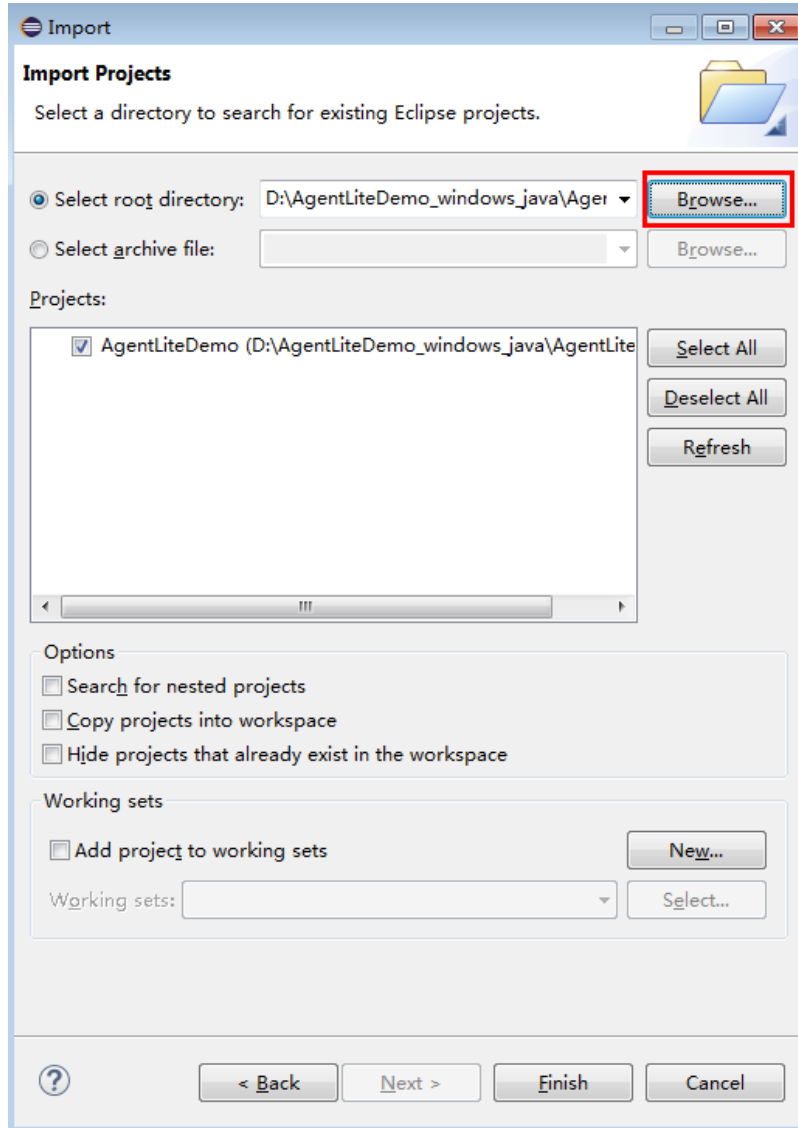
AgentLiteDemo	2018/5/9 10:03	文件夹	
RemoteSystemsTempFiles	2018/5/8 17:30	文件夹	
AgentLiteDemo.rar	2018/5/23 16:42	WinRAR archive	1,401 KB

**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

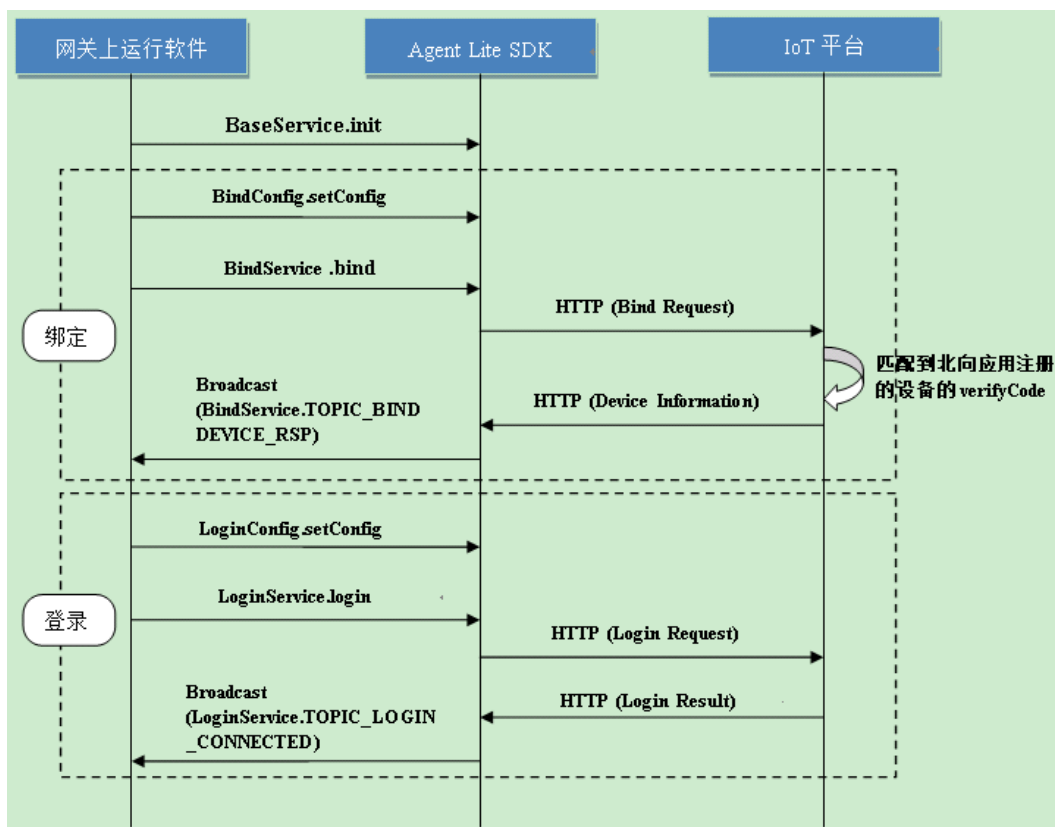
## 初始化

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);
```

## 绑定和登录



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验证码以及设备信息，将设备或网关绑定到物联网平台。

### 步骤1 修改绑定参数。

绑定时使用的设备固有信息（如设备型号等）是从“AgentLiteBind.java”文件中读取的，所以需要修改./src/main目录下“AgentLiteBind.java”文件中的如下信息：

物联网平台的设备对接地址（MQTTs）和端口，可参考[平台对接信息](#)获取。

```
//设置配置参数
private static final String PLATFORM_IP = "100.100.100.100";
private static final String HTTPS_PORT = "8943";
```

“verifyCode”（设备验证码）和必要的设备信息，包括“nodeId”（设备标识码）、“manufacturerId”（厂商Id）、“deviceType”（设备类型）、“model”（设备模型）和“protocolType”（协议类型）。“verifyCode”的值与“nodeId”保持一致，“manufacturerId”（厂商Id）、“deviceType”（设备类型）、“model”（设备模型）和“protocolType”（协议类型）与Profile文件中的定义保持一致。

### 说明

- 如果开发者通过“设备管理服务控制台”注册设备，则“verifyCode”填写为设备注册时的“preSecret”（预置密钥）。
- 如果通过开发中心注册设备，则“verifyCode”填写为设备注册时设置的“nodeId”（设备标识）。

```
public void bindAction() {
    System.out.println("===== start bind =====");
```

```
String nodeId = "1234568";
String verifyCode = "1234568";
String manufactrueId = "Huawei";
String deviceType = "Gateway";
String model = "AgentLite01";
String protocolType = "LWM2M";
deviceInfo = new IotaDeviceInfo(nodeId, manufactrueId, deviceType, model, protocolType);
...
}
```

## 步骤2 绑定设备。

注册观察者对设备绑定结果进行相应处理。

```
//注册观察者
AgentLiteBind agentLiteBind = AgentLiteBind.getInstance();
BindService bindService = BindService.getInstance();
bindService.registerObserver(agentLiteBind);
//网关绑定
agentLiteBind.bindAction();
```

调用API接口**BindConfig.setConfig()**设置绑定配置，接着调用API接口**BindService.bind(String verifyCode, IotaDeviceInfo deviceInfo)**绑定设备。

```
public void bindAction() {
    System.out.println("==== start bind =====");
    ...
    //绑定配置
    configBindPara();
    //发起绑定请求
    BindService.bind(verifyCode, deviceInfo);
}
//绑定配置
private static void configBindPara() {
    boolean res = false;
    res = BindConfig.setConfig(BindConfig.BIND_CONFIG_ADDR, PLATFORM_IP);
    res = BindConfig.setConfig(BindConfig.BIND_CONFIG_PORT, HTTPS_PORT);
    ...
}
```

设备或网关绑定成功，后续就不需要再绑定了，除非设备或网关被删除，才需要重新绑定。

设备绑定成功会收到**BindService**发出的通知，通知内容请参考Agent Lite API接口文档中设备绑定接口的返回结果说明和demo中**update**函数的处理。

## 步骤3 修改登录参数。

在demo的./src/main/AgentLiteLogin.java设置物联网平台的接入IP与端口。

```
private static final String PLATFORM_IP = "100.100.100.100";
private static final String MQTTS_PORT = "8883";
private static final String HTTPS_PORT = "8943";
```

## 步骤4 设备登录。

注册观察者对设备登录结果进行相应处理。

```
//注册观察者
AgentLiteLogin agentLiteLogin = AgentLiteLogin.getInstance();
LoginService loginService = LoginService.getInstance();
loginService.registerObserver(agentLiteLogin);
//网关登录
agentLiteLogin.loginAction();
```

调用API接口**LoginConfig.setConfig()**传入所需的登录信息，接着调用API接口**LoginService.login()**进行直连设备登录，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。

- “设备Id”（即网关Id, “LOGIN\_CONFIG\_DEVICEID”），“appId”（“LOGIN\_CONFIG\_APPID”）和“密码”（“LOGIN\_CONFIG\_SECRET”），这些信息都是从网关绑定成功的通知中得到的。
- “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。

```
public void loginAction() {
    System.out.println(" ===== start login ===== ");
    configLoginPara();
    LoginService.login();
}
private static void configLoginPara() {
    if (AgentLiteUtil.isEmpty(GatewayInfo.getDeviceID())
        || AgentLiteUtil.isEmpty(GatewayInfo.getAppID())
        || AgentLiteUtil.isEmpty(GatewayInfo.getSecret())) {
        String jsonStr = AgentLiteUtil.readToString("./workdir/gwbindinfo.json");
        JSONObject json = new Gson().fromJson(jsonStr, JSONObject.class);
        GatewayInfo.setDeviceID(json.get("deviceid").getAsString());
        GatewayInfo.setAppID(json.get("appid").getAsString());
        GatewayInfo.setSecret(json.get("deviceSecret").getAsString());
    }
    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, PLATFORM_IP);
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_IOCM_PORT, HTTPS_PORT);
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_MQTT_ADDR, PLATFORM_IP);
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_MQTT_PORT, MQTTS_PORT);
}
```

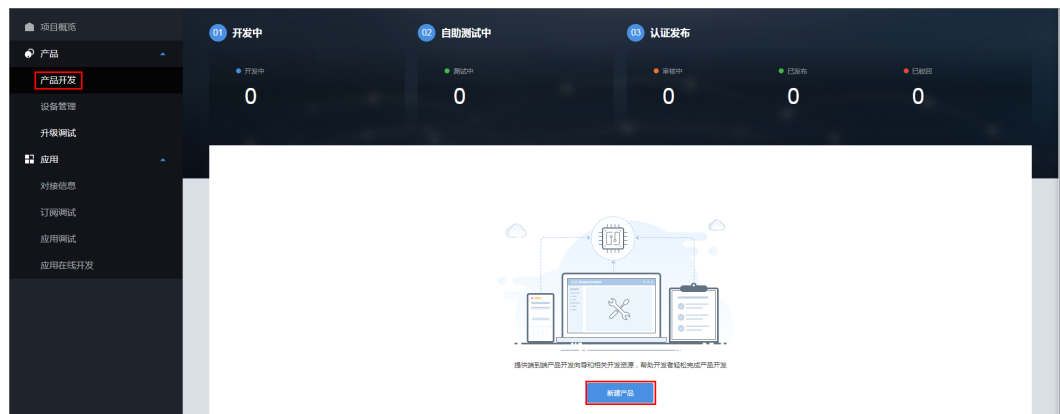
----结束

## 上传 Profile 并注册设备

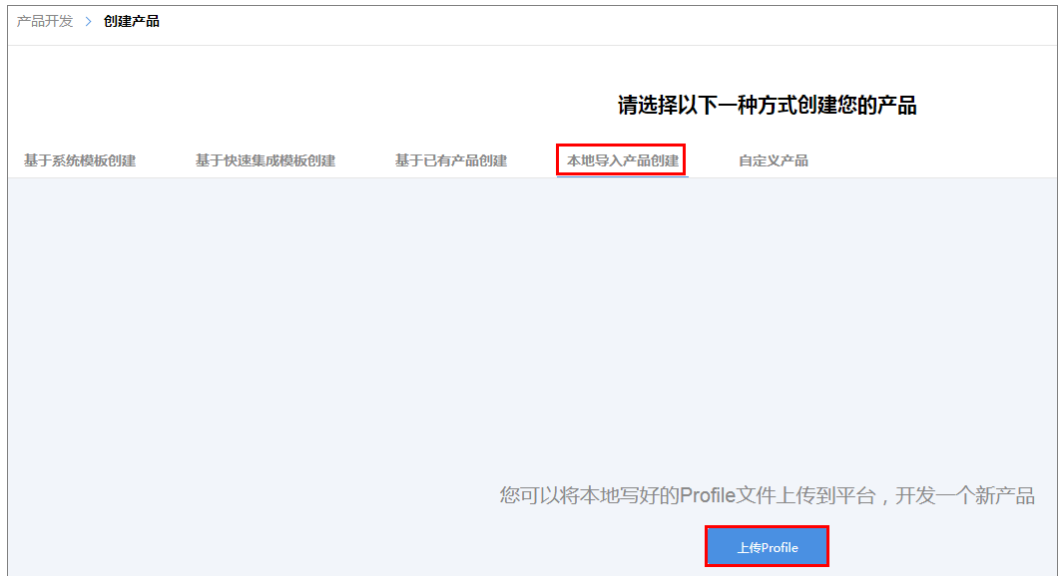
下载[Profile开发示例](#)，并上传模板中的profile文件：

“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

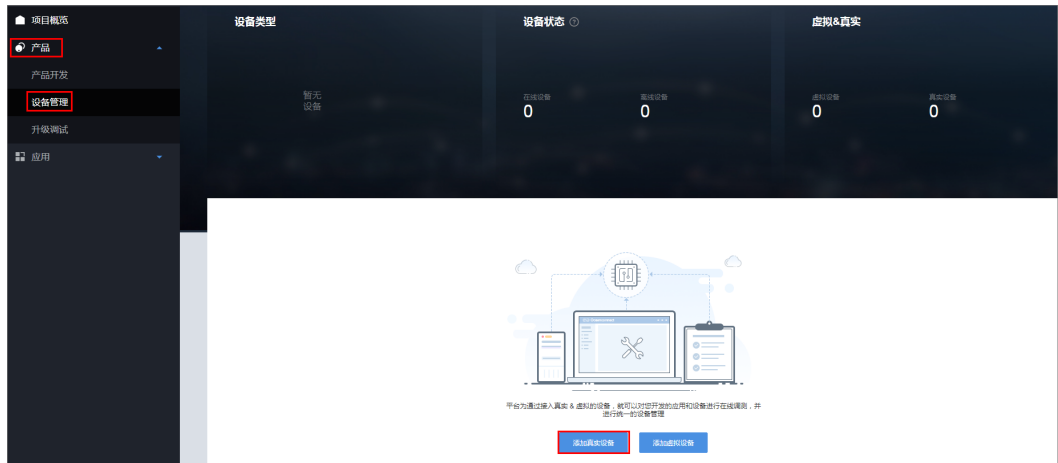
**步骤1** 登录开发中心，创建一个项目，在该项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“创建产品”中，选择“本地导入产品创建”，单击“上传Profile”上传“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

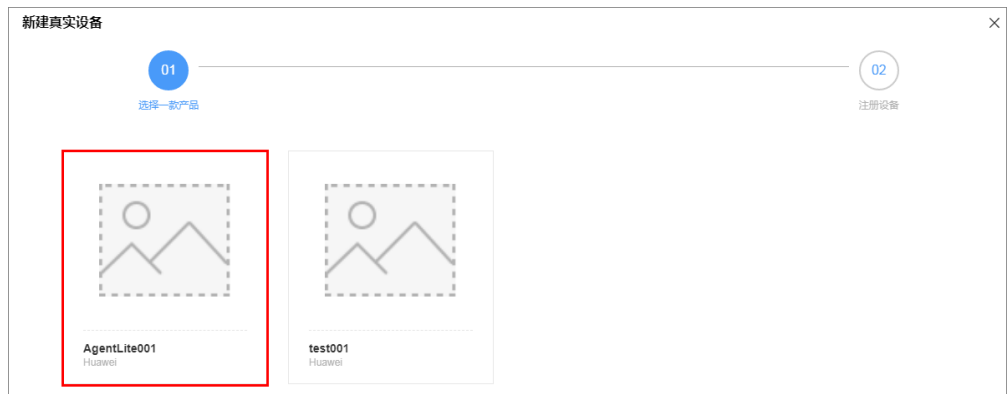


**步骤3** 选择“产品 > 设备管理”，单击“添加真实设备”，进入“新建增实设备”页面。



**步骤4** 根据向导注册设备。

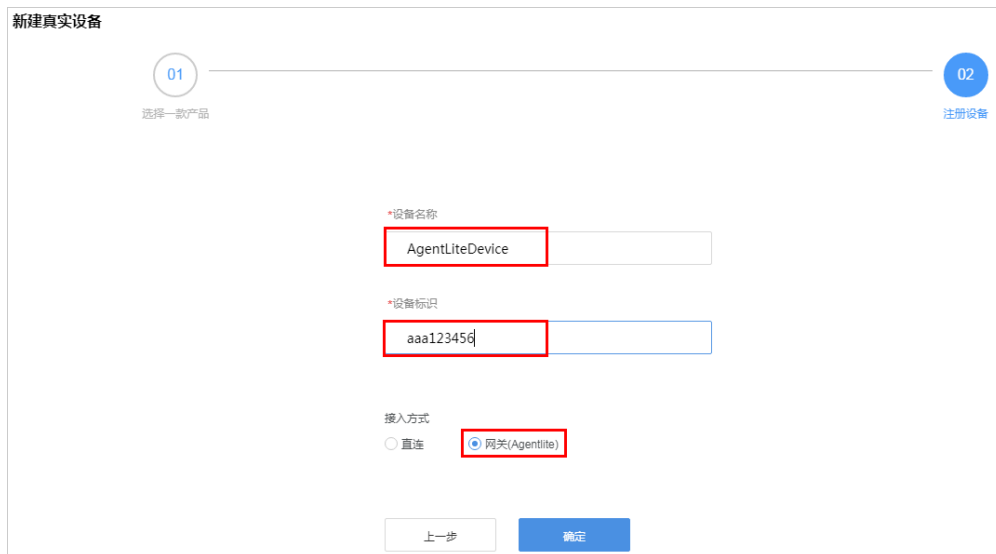
1. 选择产品。  
产品：AgentLite001



2. 填写设备相关信息，单击“确定”。
  - 设备名称：AgentLiteDevice



- 设备标识: aaa123456, 需要与AgentLiteDemo中网关的设备标识一致。
- 接入方式: 网关 (Agentlite)



**步骤5** 注册设备后, Agent Lite SDK发送bind消息, 则在开发中心的“产品 > 设备管理 > 设备列表”界面可以看到设备状态变成“在线”。



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。

----结束

## 数据上报和数据发布

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: 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。注册观察者对网关数据上报结果进行相应处理。

```
//注册观察者
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();
DataTransService dataTransService = DataTransService.getInstance();
dataTransService.registerObserver(agentLiteDataTrans);
//数据上报
agentLiteDataTrans.gwDataReport();
```

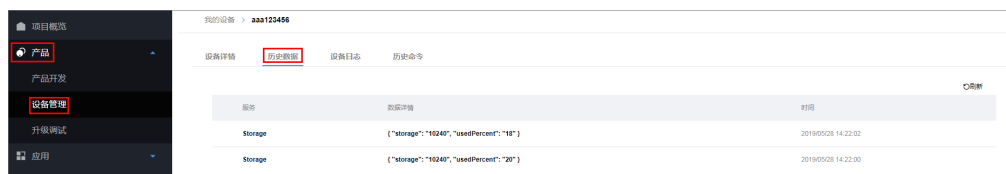
设备或网关登录成功后可以调用**DataTransService.dataReport(int cookie, String reqstId, String deviceId, String serviceId, String serviceProperties)**接口上报数据。

```
public void gwDataReport() {
    System.out.println(" ===== gwDataReport! ===== ");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);

    String deviceId = GatewayInfo.getDeviceId();

    JsonObject data = new JsonObject();
    data.addProperty("storage", "10240");
    data.addProperty("usedPercent", "20");
    DataTransService.dataReport(cookie, null, deviceId, "Storage", data.toString());
}
```

数据上报成功后可以在设备的“历史数据”中看到上报的数据了。



- “数据发布”接口：topic固定为“/cloud/signaltrans/v2/categories/data”；“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。

注册观察者对网关数据上报结果进行相应处理。

```
//注册观察者
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();
DataTransService dataTransService = DataTransService.getInstance();
dataTransService.registerObserver(agentLiteDataTrans);
//数据发布
agentLiteDataTrans.gwDataReportByMqttDataPub();
```

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。

- “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。

```
public void gwDataReportByMqttDataPub() {
    System.out.println(" ===== gwDataReportByMqttDataPub! ===== ");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);

    String deviceId = GatewayInfo.getDeviceId();

    JsonObject headerData = new JsonObject();
    headerData.addProperty("method", "PUT");
    String fromStr = "/device/"+deviceId+"/services/Storage";
    String toStr = "/data/v1.1.0/devices/"+deviceId+"/services/Storage";
    headerData.addProperty("from", fromStr);
    headerData.addProperty("to", toStr);

    headerData.addProperty("access_token", GatewayInfo.getAccessToken());

    SimpleDateFormat df = new SimpleDateFormat(MSG_TIMESTAMP_FORMAT);
    df.setTimeZone(TimeZone.getTimeZone("UTC"));
```

```
String curTime = df.format(new Date(System.currentTimeMillis()));
headerData.addProperty("timestamp", curTime);
headerData.addProperty("eventTime", curTime);

JsonObject bodyData = new JsonObject();
bodyData.addProperty("storage", "10240");
bodyData.addProperty("usedPercent", "18");

JsonObject mqttMsg = new JsonObject();
mqttMsg.add("header", headerData);
mqttMsg.add("body", bodyData);

DataTransService.mqttDataPub(cookie, "/cloud/signaltrans/v2/categories/data", 1,
mqttMsg.toString().getBytes());
}
```

## 命令接收

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。

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。

注册观察者对命令接收进行相应处理。

```
//注册观察者
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();
DataTransService dataTransService = DataTransService.getInstance();
dataTransService.registerObserver(agentLiteDataTrans);
//命令接收
agentLiteDataTrans.getCmdReceive();
```

被动接收命令的方法getCmdReceive：

```
private void getCmdReceive(IotaMessage iotaMsg) {
    System.out.println("=====receive iotCMD =====");
    String deviceId = iotaMsg.getString(DataTransService.DATATRANS_IE_DEVICEID);
    String requestId = iotaMsg.getString(DataTransService.DATATRANS_IE_REQUESTID);
    String serviceId = iotaMsg.getString(DataTransService.DATATRANS_IE_SERVICEID);
    String method = iotaMsg.getString(DataTransService.DATATRANS_IE_METHOD);
    String cmd = iotaMsg.getString(DataTransService.DATATRANS_IE_CMDCONTENT);
    if (method.equals("REMOVE_GATEWAY")) {
        //rmvGateway(context);
    }
    System.out.println ("Receive cmd :"+
        + "ndeviceId = " + deviceId
        + "nrequestId = " + requestId
        + "nserviceId = " + serviceId
        + "nmethod = " + method
        + "ncmd = " + cmd);
}
```

## 添加非直连设备

在添加非直连设备前，确认非直连设备的profile已经上传了，详见[上传Profile并注册设备](#)步骤。

修改非直连设备信息，包括“nodeId”（设备标识码）、“manufactureId”（厂商Id）、“deviceType”（设备类型）、“model”（设备模型）和“protocolType”（协议类型）。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。

```
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);
...
}
```

注册观察者对添加设备结果进行相应处理。

```
//注册观察者
AgentLiteHub agentLiteHub = AgentLiteHub.getInstance();
HubService hubService = HubService.getInstance();
hubService.registerObserver(agentLiteHub);
//sensor添加
agentLiteHub.addSensor();
```

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。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”。

```
public void addSensor() {
    System.out.println("===== addSensor! ===== ");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);
    ...
    HubService.addDevice(cookie, deviceInfo);
}
```

非直连设备添加成功后在“设备列表”中看到新增一条记录。



## 非直连设备状态更新

注册观察者对非直连设备状态更新结果进行相应处理。

```
//注册观察者
AgentLiteHub agentLiteHub = AgentLiteHub.getInstance();
HubService hubService = HubService.getInstance();
hubService.registerObserver(agentLiteHub);
//sensor状态更新
agentLiteHub.updateDeviceStatus();
```

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 updataDeviceStatus() {
    System.out.println(" ===== updataDeviceStatus ===== ");
    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, "ONLINE", "NONE");
}
```

## 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。

设备数据上报成功后，可以在非直连设备的“历史数据”中查看上报的数据。



### 4.6.1.4.3 Agent Lite SDK 使用指南（Android）（联通用户专用）

非联通用户请查看[设备接入服务](#)。

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 Android 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 Android Demo开发，也可参考Agent Lite Android Demo，自行集成Agent Lite SDK(Android)。
- Agent Lite Android Demo使用的IDE工具为**Android Studio**。

## 使用必读

**开发环境要求：**

Android系统：API\_LEVEL21及以上。

**工程目录结构及文件说明：**

目录结构	目录	说明
Agent LiteDemo	src	存放Agent Lite Demo代码
—src	libs	存放Agent Lite提供的jar包和第三方jar包
—com		
—huawei		

目录结构	目录	说明
└─ agentlitedemo	armeabi	存放Agent Lite编译后的库文件和第三方库文件
└─ gen		
└─ assets	conf	存放TLS证书文件、配置文件
└─ conf		
└─ bin		
└─ libs		
└─ armeabi		
└─ res		

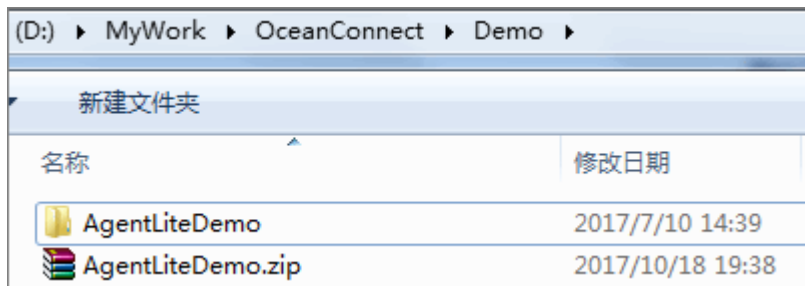
### 说明

如果开发者没有设备，可以直接在X86 Linux系统进行开发。

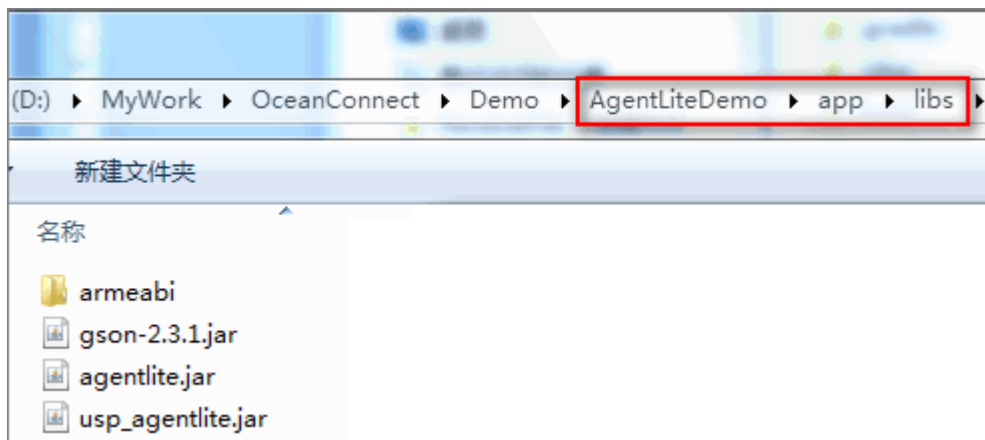
## 导入样例代码及配置 AndroidStudio

导入样例代码：

**步骤1** 将Agent Lite Demo(Android)解压到本地。



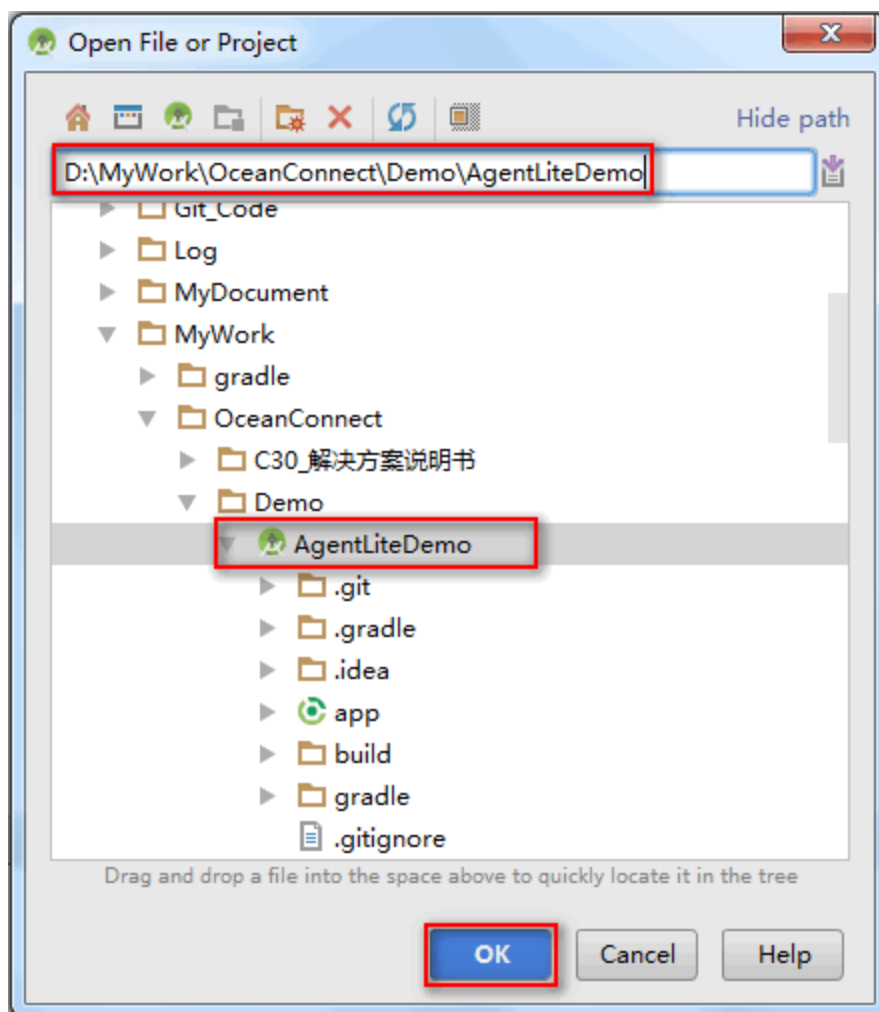
SDK在Demo中libs文件中。



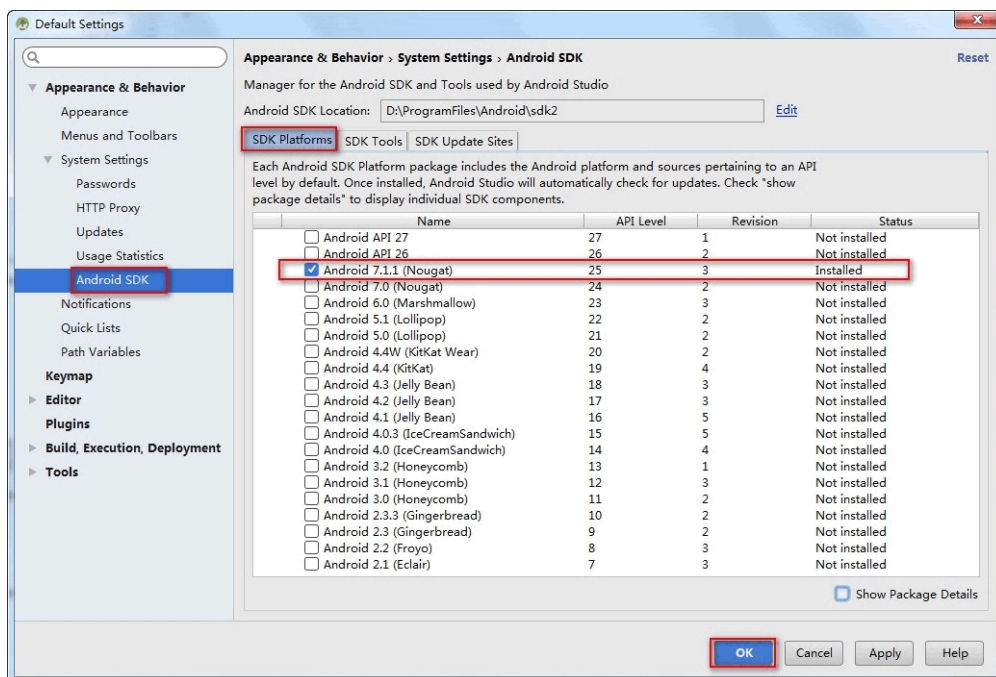
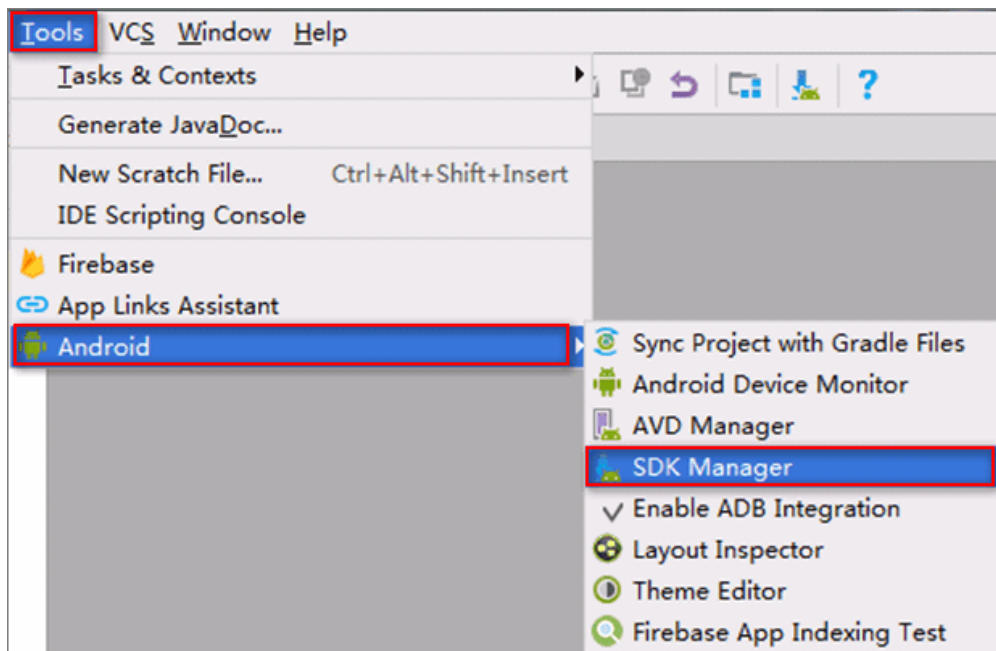
**步骤2** 导入android工程。

**注：**请确保本地计算机网络正常，才能下载需要的SDK和gradle包等资源。

- 打开Android Studio，在“file”菜单中选择“open”。
- 在打开的界面中输入AgentLiteDemo所在的路径，选择“AgentLiteDemo”，点击“OK”。

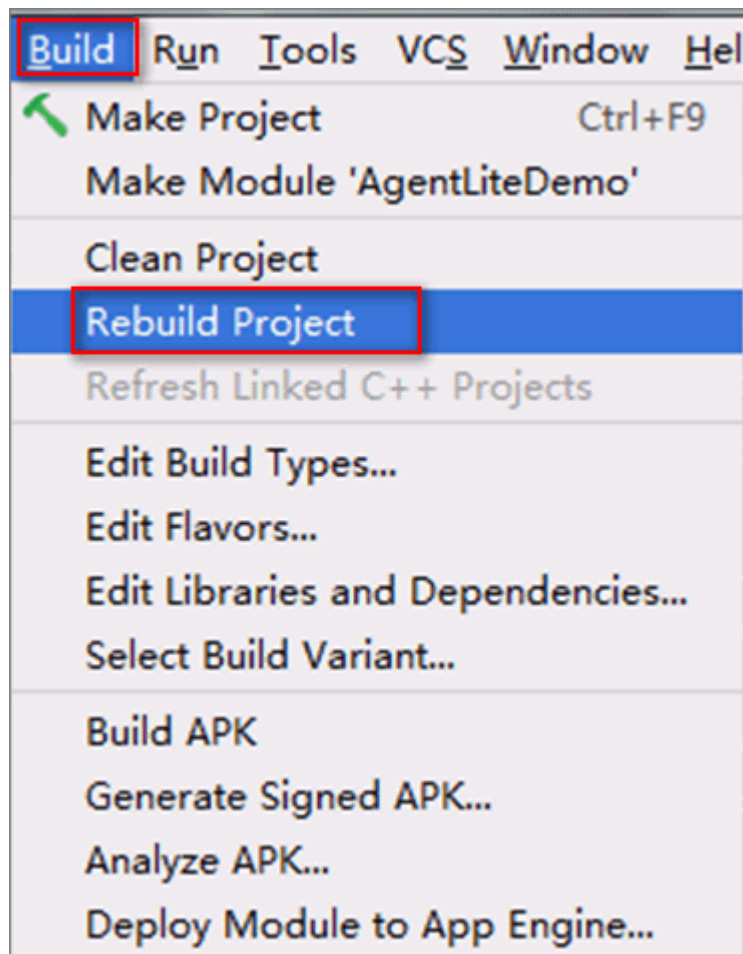


- 打开菜单“Tools > Android > SDK Manager”，在弹出的对话框中选择“Android 7.1.1(Nougat)”。



**步骤3** 等待SDK下载安装完毕，重新编译工程，选择菜单“Build > Rebuild Project”即可。如果编译过程出现问题，需要修改AndroidStudio配置，请参考下一章节的内容。





----结束

#### 配置AndroidStudio:

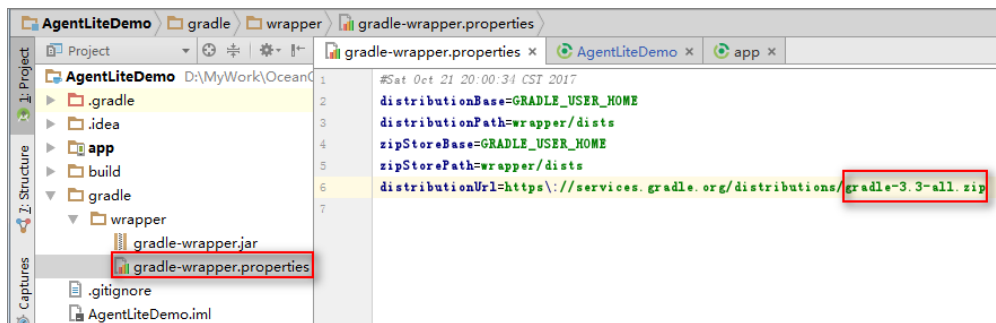
推荐使用AndroidStudio 2.X.X版本，如2.3.1。推荐理由：版本相对稳定，下载插件时不会出现后续版本的各种问题，各版本在功能方面实际上差不多。

#### 步骤1 设置Gradle和Plugin版本。

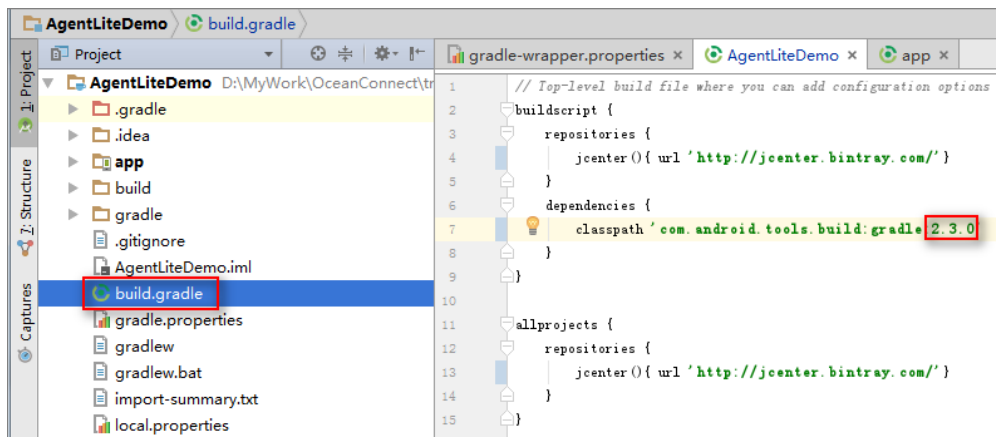
Gradle和Plugin的版本之间有配套关系，请参考：<https://developer.android.com/studio/releases/gradle-plugin>。

设置Gradle和Plugin版本时，需要注意在<https://services.gradle.org/distributions/>和<http://jcenter.bintray.com/com/android/tools/build/gradle/>目录下是否有所设置的版本（可在浏览器中打开两个网址查看）。

- 打开\gradle\wrapper目录下的gradle-wrapper.properties，设置Gradle版本。

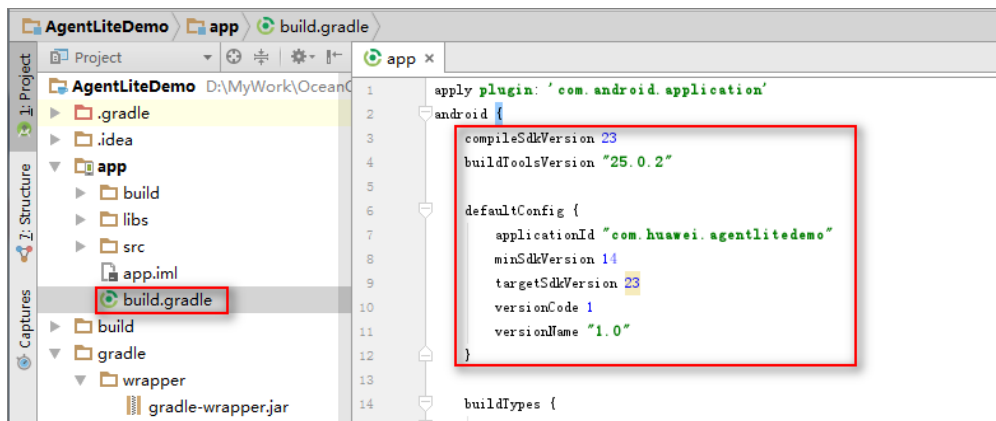


- 打开工程目录下的build.gradle，设置Plugin版本。

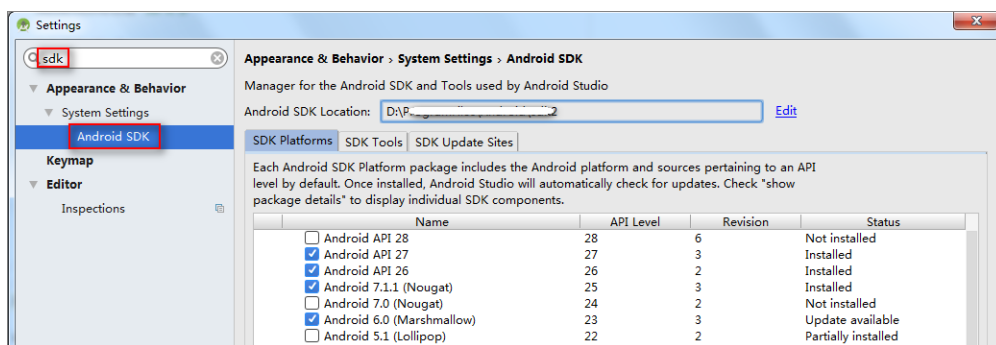


**步骤2 设置Android SDK版本。**

- 可以打开app目录下的build.gradle根据需要修改SDK版本号。

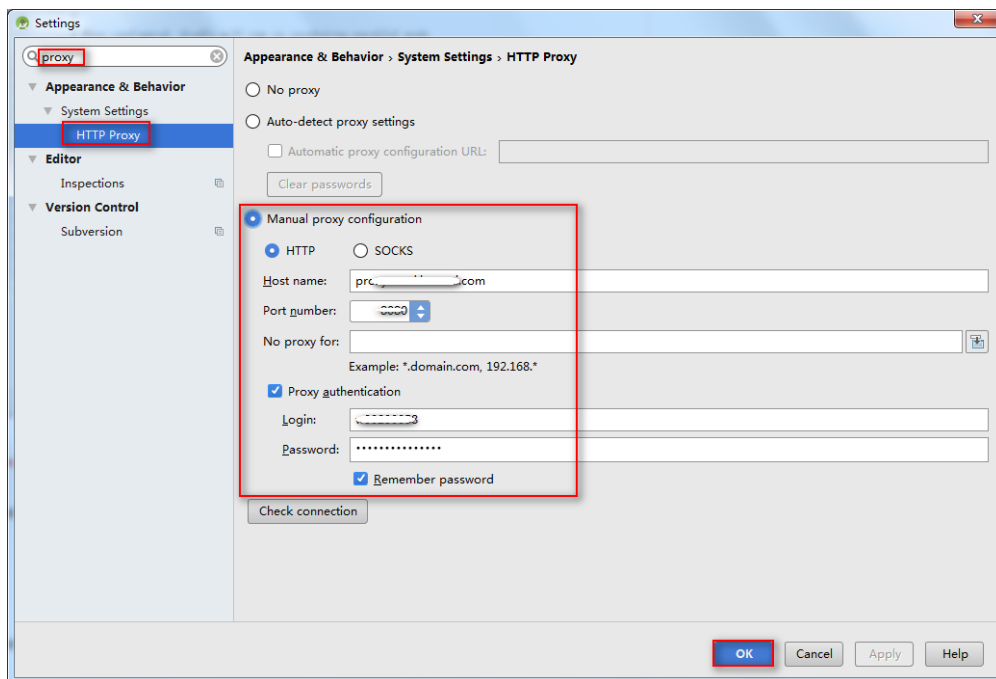


- SDK版本号修改后需要注意是否已安装对应版本的Android SDK，可以在“File > Settings > Android SDK > SDK Platforms”中查看并下载。

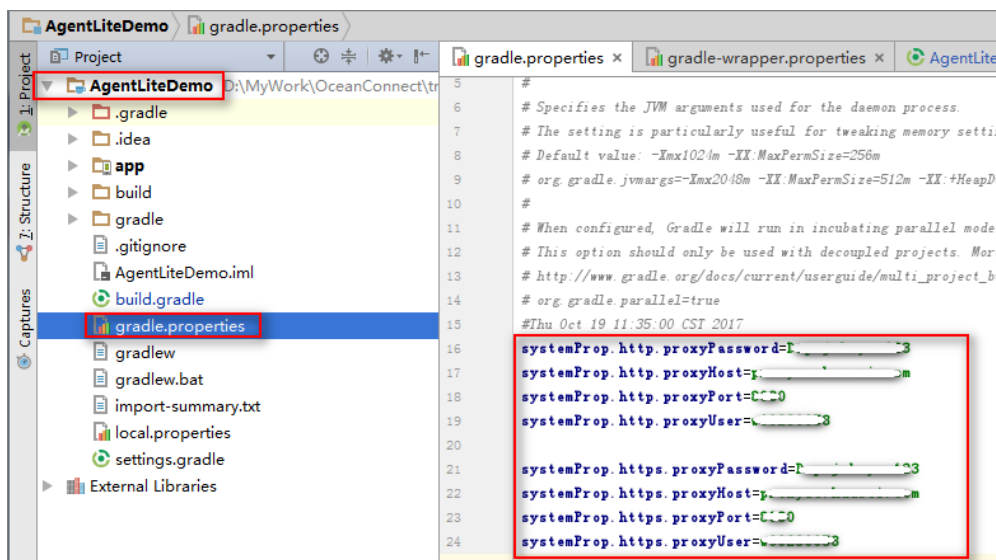


### 步骤3 代理配置。

- 打开“File > Settings > HTTP Proxy”设置好相关配置即可（如下图所示）。如果不需要使用代理，请选择No proxy。



- 打开工程目录下的gradle.properties，设置工程的http和https代理配置（一般两个配置是一样的）。



----结束

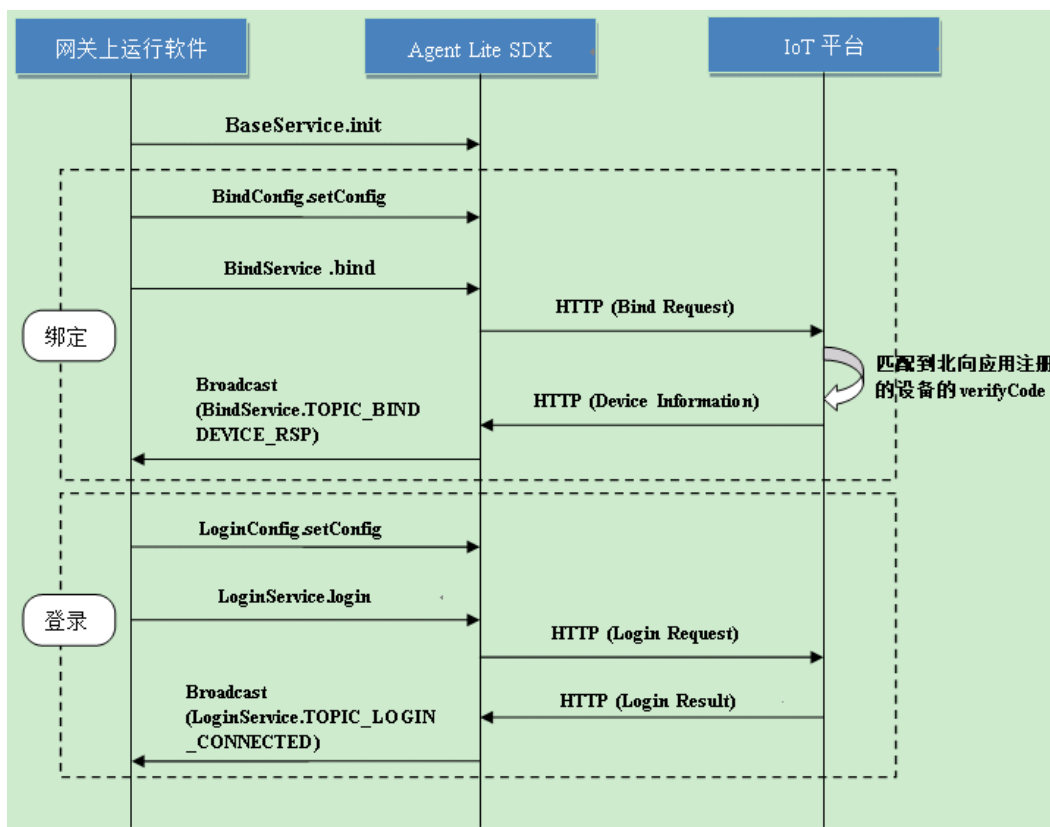
## 初始化

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。可参考“AgentLiteMain.java”中onCreate()方法对BaseService.init()的调用。

```
BaseService.init(WORK_PATH, LOG_PATH, context);
```

- “WORK\_PATH” 为工作路径，不能为空。
- “LOG\_PATH” 为打印日志路径，当“LOG\_PATH” 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。

## 绑定和登录



设备或网关第一次接入IoT联接管理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。

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。

### 步骤1 修改绑定参数。

绑定时使用的设备固有信息（如设备型号等）是从“config.properties”文件中读取的，所以需要修改./app/src/main/assets/conf目录下config.properties文件中的如下信息：

- “platformIP”：物联网平台的设备对接地址（MQTTs），可参考[平台对接信息](#)获取。
- “verifyCode”（设备的标识）和必要的设备信息，包括“Manufacture”（厂商Id）、“deviceType”（设备类型）、“HardwareModel”（设备模型）和“protocolType”（协议类型），其中“manufacturerId”（厂商Id）、“deviceType”（设备类型）、“HardwareModel”（设备模型）和“protocolType”（协议类型）与Profile文件中的定义保持一致。

## 说明

- 如果通过“设备管理服务控制台”注册设备，则“verifyCode”填写为设备注册时的“preSecret”（预置密钥）。
- 如果通过开发中心注册设备，则“verifyCode”填写为设备注册后返回的“nodeId”（设备标识）。

“config.properties”文件中设备固有示例：

```
platformIP=100.100.100.100
mqttPort=8883
httpPort=8943
verifyCode=0123456789
Manufacturer=Huawei
HardwareModel=AgentLite01
deviceType=Gateway
protocolType=HuaweiM2M
```

## 步骤2 绑定设备。

调用API接口`BindConfig.setConfig()`设置绑定配置。

```
private void startBind() {
    ....
    configBindPara();
    registerBindReceiver();
    ...
}
private void configBindPara() {
    BindConfig.setConfig(BindConfig.BIND_CONFIG_ADDR, AgentLiteUtil.get(ConfigName.platformIP));
    BindConfig.setConfig(BindConfig.BIND_CONFIG_PORT, AgentLiteUtil.get(ConfigName.httpPort));
}
```

注册广播接收器对设备绑定结果进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(mBindStatusReceiver, new
IntentFilter(BindService.TOPIC_BINDDEVICE_RSP));
```

调用API接口`BindService.bind(String verifyCode, IotaDeviceInfo deviceInfo)`绑定设备，主要入参为“verifyCode”（设备验证码）和必要的设备信息，包括“nodeId”（设备标识码）、“Manufacturer”（厂商Id）、“deviceType”（设备类型）、“HardwareModel”（设备模型）和“protocolType”（协议类型），其中“verifyCode”的值与“nodeId”保持一致。

```
private void startBind() {
    LogUtil.i(this, TAG, "startBind");
    String nodeId = AgentLiteUtil.get(ConfigName.verifyCode);
    String verifyCode = AgentLiteUtil.get(ConfigName.verifyCode);
    String manufacturerId = AgentLiteUtil.get(ConfigName.Manufacturer);
    String deviceType = AgentLiteUtil.get(ConfigName.deviceType);
    String model = AgentLiteUtil.get(ConfigName.HardwareModel);
    String protocolType = AgentLiteUtil.get(ConfigName.protocolType);

    IotaDeviceInfo deviceInfo = new IotaDeviceInfo(nodeId, manufacturerId, deviceType, model,
protocolType);

    ...

    BindService.bind(verifyCode, deviceInfo);
}
```

设备或网关绑定成功，后续就不需要再绑定了，除非设备或网关被删除，才需要重新绑定。

设备绑定成功会收到广播，广播内容请参考Agent Lite API接口文档中设备绑定接口的返回结果说明和demo中`mBindStatusReceiver`函数的处理。

### 步骤3 配置登录参数。

登录前需要通过参数配置接口**LoginConfig.setConfig()**传入所需的登录信息。

- “设备Id”（即网关Id, “LOGIN\_CONFIG\_DEVICEID”），“appId”（“LOGIN\_CONFIG\_APPID”）和“密码”（“LOGIN\_CONFIG\_SECRET”），这些信息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**mBindStatusReceiver**函数的处理。

```
private 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, 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_PORT, "8883");  
}
```

### 步骤4 设备登录。

注册广播接收器对设备登录结果进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(LoginConnectReceiver, new  
IntentFilter(LoginService.TOPIC_LOGIN_CONNECTED));
```


调用**LoginService.login()**进行直连设备登录，具体API的参数使用参考Agent Lite接口文档中设备登录接口的说明。

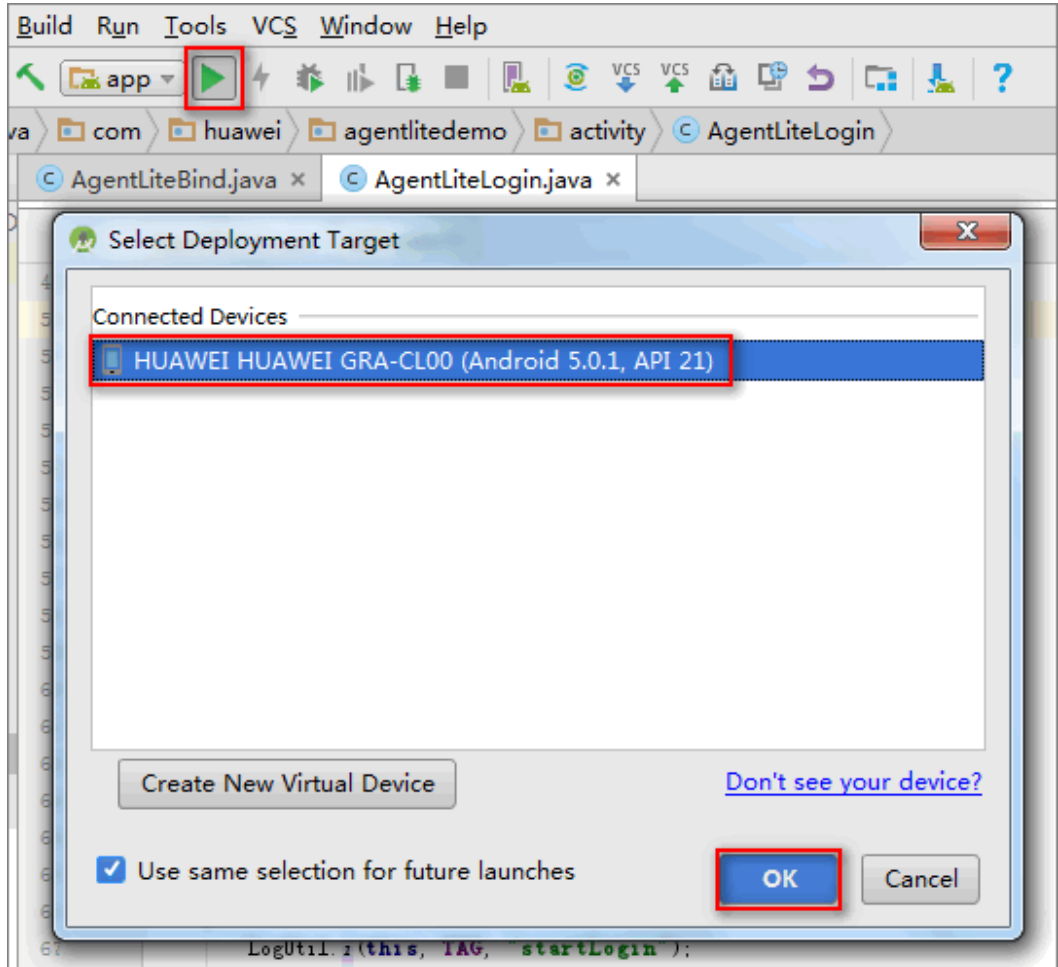
```
private void startLogin() {  
    LogUtil.i(this, TAG, "startLogin");  
    configLoginPara();  
    LoginService.login();  
}
```

----结束

## 编译安装程序

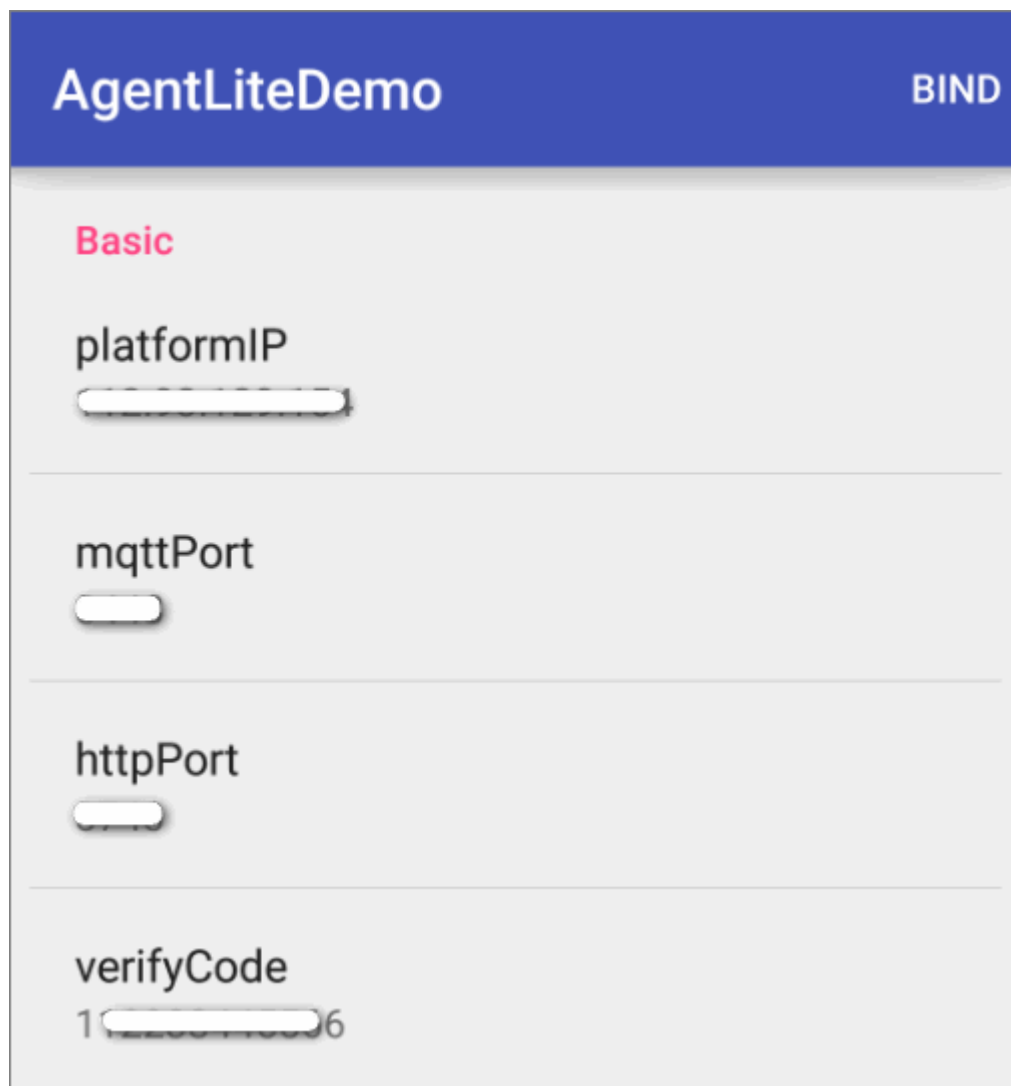
绑定和登录功能完成后，可以先测试一下网关是否能正常与平台对接，再进行后续功能开发。

**步骤1** 将Android设备连到计算机上，点击“”按钮编译安装工程到Android设备上。



**步骤2** 在Android设备上运行AgentLiteDemo，修改“平台IP”、“MQTT端口”、“HTTP端口”和“verifyCode”等值。

- “平台IP”、“MQTT端口”和“HTTP端口”可以从物联网平台环境开发中心的“对接信息”界面获取。
- “verifyCode”则是设备的标识，每个设备对应一个verifyCode，不可重复，所以建议使用IMEI或者MAC地址等天然的设备标识。AgentLiteDemo可以在界面上人工输入设备的verifyCode，测试时只要输入一个没有使用过的verifyCode即可。



----结束

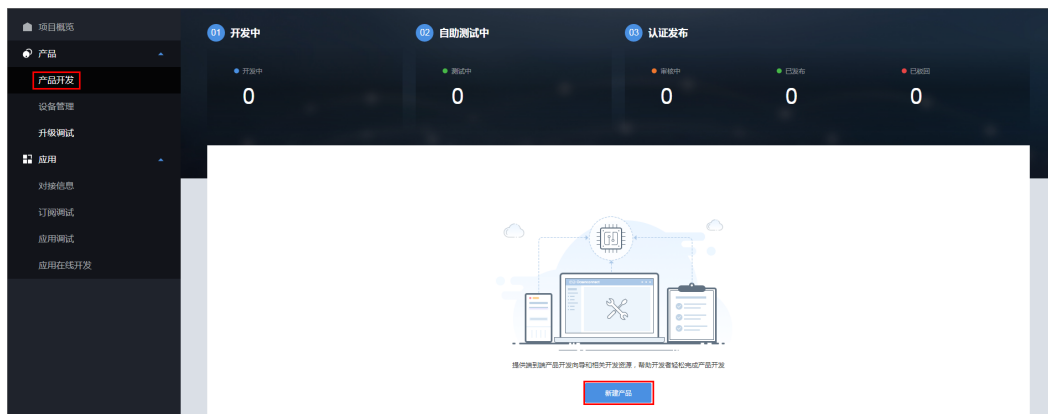
## 上传 Profile 并注册设备

下载[Profile开发示例](#)，并上传模板中的profile文件：

“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

**步骤1** 登录开发中心，创建一个项目，在该项目空间内，选择“产品 > 产品开发”，点击“新建产品”。

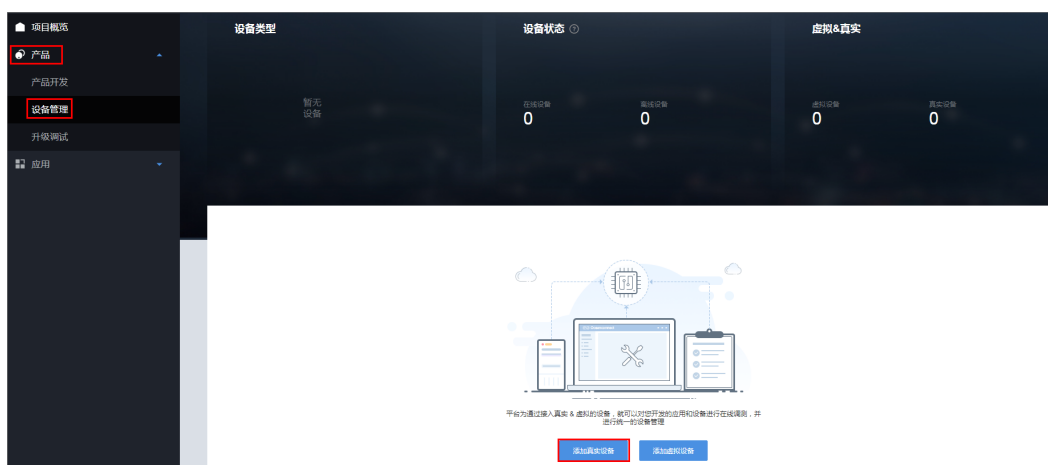




**步骤2** 在“创建产品”中，选择“本地导入产品创建”，单击“上传Profile”上传“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

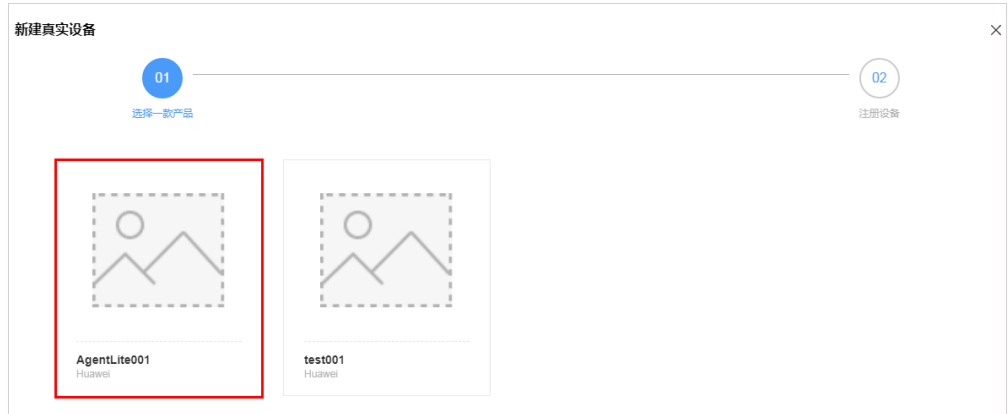


**步骤3** 选择“产品 > 设备管理”，单击“添加真实设备”，进入“新建增实设备”页面。

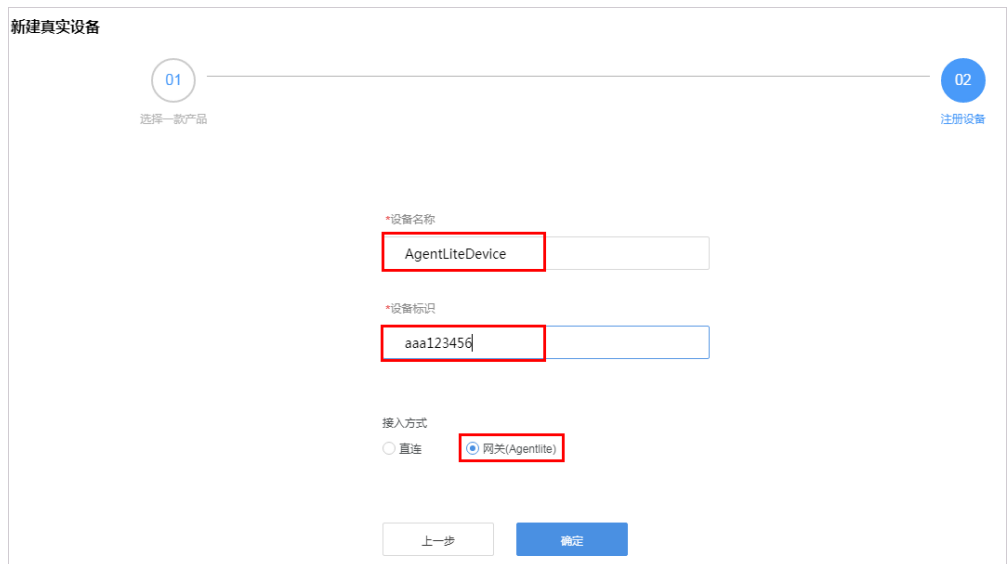


**步骤4** 根据向导注册设备。

1. 选择产品。  
产品：AgentLite001



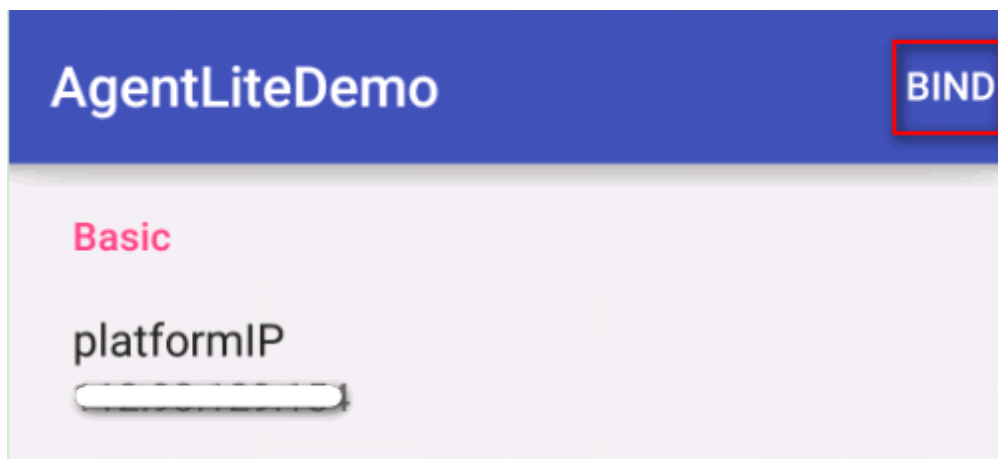
2. 填写设备相关信息，单击“确定”。
  - 设备名称：AgentLiteDevice
  - 设备标识：aaa123456，需要与AgentLiteDemo中网关的设备标识一致。
  - 接入方式：网关（Agentlite）



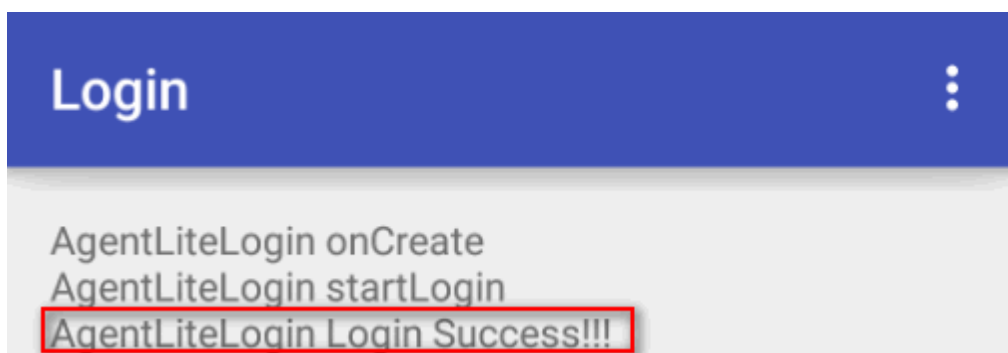
----结束

## 设备上线

- 步骤1 点击AgentLiteDemo上的“BIND”按钮，查看界面上打印出的“设备日志”。



步骤2 当出现“AgentLiteLogin Login Success”，表示AgentLite登录平台成功。



----结束

## 数据上报和数据发布

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：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。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。
  - 当设备主动上报数据时，“requestId”可以为空。
  - 当上报的数据为某个命令的响应时，“requestId”必须与下发命令中的“requestId”保持一致。requestId可以从广播中获取，请参考API文档中“设备命令接收”接口的广播参数“DATATRANS\_IE\_REQUESTID”的说明。
  - “serviceId”要与profile中定义的某个serviceId保持一致，否则无法上报数据。
  - “serviceProperties”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。

```
private void gatewayDataReport() {  
    LogUtil.i(this, TAG, "gatewayDataReport!");  
    int cookie;  
    String deviceId = GatewayInfo.getDeviceID();
```

```
Random random = new Random();
cookie = random.nextInt(65535);
LogUtil.i(this, TAG, "cookie = " + cookie);
DataTransService.dataReport(cookie, null, deviceId, "Storage", "{\\"storage\\":10240,\\"usedPercent\\":20}");
}
```

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(dataReportRsp, new
IntentFilter(DataTransService.TOPIC_DATA_REPORT_RSP));
```

- “数据发布”接口：topic固定为“/cloud/signaltrans/v2/categories/data”；“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。

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。

- “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。

```
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));
```

## 命令接收

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。

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**DataTransService.TOPIC\_COMMAND\_RECEIVE**广播接收器对命令下发进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(commandReceiver, new
IntentFilter(DataTransService.TOPIC_COMMAND_RECEIVE));
```

具体的处理函数**commandReceiver**：

```
private BroadcastReceiver commandReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        IotaMessage iotaMsg = (IotaMessage)
intent.getSerializableExtra(DataTransService.DATATRANS_BROADCAST_IE_IOTAMSG);
        String deviceId = iotaMsg.getString(DataTransService.DATATRANS_IE_DEVICEID);
        String requestId = iotaMsg.getString(DataTransService.DATATRANS_IE_REQUESTID);
        String serviceId = iotaMsg.getString(DataTransService.DATATRANS_IE_SERVICEID);
        String method = iotaMsg.getString(DataTransService.DATATRANS_IE_METHOD);
        String cmd = iotaMsg.getString(DataTransService.DATATRANS_IE_CMDCONTENT);

        if (method.equals("REMOVE") && deviceId.equals(sensorId)) {
            //do something, e.g. show a dialog
            //rmvSensor();
        }
    }
}
```

```

LogUtil.i(AgentLiteLogin.this, TAG, "Receive cmd :"+
+ "\ndeviceId = " + deviceId + "\nrequestId = " + requestId
+ "\nserviceName = " + serviceName + "\nmethod = " + method
+ "\ncommand = " + cmd);
}
};
    
```

在开发中心的“产品 > 设备管理”界面中，单击非直连设备列后的“删除”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。

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将设备删除。

## 添加非直连设备

在添加非直连设备前，确认非直连设备的profile已经上传了，详见[上传Profile并注册设备](#)步骤。

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。

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。

```

private void addSensor() {
    SharedPreferences preferences = getSharedPreferences("AgentLiteDemo", MODE_PRIVATE);
    if (preferences.getString("SENSORID", null) != null) {
        Toast.makeText(this, "The sensor is already added.", Toast.LENGTH_SHORT).show();
        return;
    }
    LogUtil.i(this, TAG, "addSensor!");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);

    IotaDeviceInfo deviceInfo = new IotaDeviceInfo("0123456test", "Huawei", "Motion", "test01", "MQTT");
    HubService.addDevice(cookie, deviceInfo);
}
    
```

注册广播接收器对添加设备结果进行相应处理。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”。

```

LocalBroadcastManager.getInstance(this).registerReceiver(addDeviceReceiver, new
IntentFilter(HubService.TOPIC_ADDDEV_RSP));
    
```

非直连设备添加成功后可以在“设备列表”中看到新增一条记录。



## 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。

```
private void updateDeviceStatus(String status, String statusDetail) {
    LogUtil.i(this, TAG, "updateDeviceStatus!");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);

    SharedPreferences preferences = getSharedPreferences("AgentLiteDemo", MODE_PRIVATE);
    String deviceId = preferences.getString("SENSORID", null);

    if (deviceId != null) {
        HubService.updateDeviceStatus(cookie, deviceId, status, statusDetail);
    }
}
```

AgentLiteDemo中只添加了一个非直连设备，所以**updateDeviceStatus()**方法中使用的“deviceId”是直接从SharedPreferences中读取的。

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。

```
LocalBroadcastManager.getInstance(this).registerReceiver(devStatusUpdateReceiver, new
IntentFilter(HubService.TOPIC_DEVSTATUS_RSP));
```

## 非直连设备数据上报

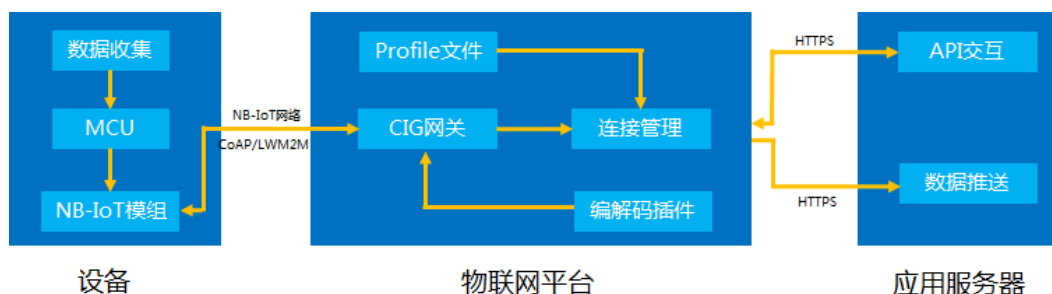
请参考[数据上报和数据发布](#)章节，调用**DataTransService.dataReport**或**DataTransService.mqttDataPub**接口进行数据上报，各个参数使用非直连设备的相关数据即可，此处不再复述。

设备数据上报成功后，可以在非直连设备的“历史数据”中查看上报的数据。



### 4.6.1.5 使用模组接入

集成NB-IoT模组的设备，可以通过NB-IoT网络接入物联网平台。



<b>特点</b>	<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>
<b>应用场景</b>	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。
<b>网络需求</b>	<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：向模组厂商购买已认证模组。您可以访问<a href="#">华为云市场</a>，购买已认证的模组。</li> </ul>
<b>物联网接入协议</b>	CoAP/LWM2M
<b>相关资源</b>	请从模组厂商获取更多信息和支撑。

## AT 指令集

AT指令用于控制设备。如下AT命令仅供参考，具体命令集请向相应的模组厂商获取。

AT命令	作用	备注
AT+CMEE=1	报错查询。	标准AT指令
AT+CFUN=0	关机。设置IMEI和平台IP端口前要先关机。	标准AT指令
AT+CGSN=1	查询IMEI。IMEI为设备标识，应用服务器调用API接口注册设备时，nodeId/verifyCode都需要设置为IMEI。	标准AT指令
AT+NTSETID=1,xxxx	xxxx为IMEI。如果查询不到可自行设置IMEI，IMEI必须是唯一的，不能与其他设备重复，且只能设置一次。 IMEI为设备标识，应用服务器调用API接口注册设备时，如果设备使用海思芯片，则nodeId/verifyCode都需要设置成IMEI；如果设备使用高通芯片，则nodeId/verifyCode都需要设置成urn:imei:IMEI。	海思芯片私有AT指令，在flash中保存IMEI。应用服务器在向平台进行设备注册时，使用此参数，其他芯片或模组厂商可参考实现。

AT命令	作用	备注
AT+NCDP="IP","port"	设置设备对接的物联网平台的IP地址和端口号，5683为非加密端口，5684为DTLS加密端口。	海思芯片私有AT指令，在flash中保存IP和端口。应用服务器在向平台进行设备注册时，使用此参数，其他芯片或模组厂商可参考实现。
AT+CFUN=1	开机。	标准AT指令
AT+NBAND=频段	设置频段。	海思芯片私有AT指令，在flash中保存频段。设备在入网时，使用此参数，其他芯片或模组厂商可参考实现。
AT+CGDCONT=1,"IP","CTNB"	设置核心网APN。APN与设备的休眠、保活等模式有关，需要与运营商确认。	标准AT指令
AT+CGATT=1	入网。	标准AT指令
AT+CGPADDR	获取终端IP地址。	标准AT指令
AT+NMGS=x,xxxx	发送上行数据。第1个参数为字节数，第2个参数为上报的16进制业务码流。	海思芯片私有AT指令，初次发送数据时，完成设备注册；后续发送数据时，仅发送数据。其他芯片或模组厂商可参考实现。
AT+NQMGR	接收下行数据。	海思芯片私有AT指令，查询接收buffer中可以接收的数据量，以及当前总共接收的消息数和丢弃的消息数。其他芯片或模组厂商可参考实现。
AT+NMGR	读取数据。	海思芯片私有AT指令，读取从物联网平台(LWM2M SERVER)接收到的数据。其他芯片或模组厂商可参考实现。



### 4.6.1.5.1 Agent Tiny SDK 使用指南（联通用户专用）

非联通用户请查看[设备接入服务](#)。

Agent Tiny是部署在具备广域网能力、对功耗/存储/计算资源有苛刻限制的终端设备上的轻量级互联互通中间件，您只需调用API接口，便可实现设备快速接入到物联网平台以及数据上报和命令接收等功能，相关集成指导请参见[端云互通组件开发指南](#)。

### 4.6.1.6 软固件升级调测

#### 4.6.1.6.1 固件升级（联通用户专用）

非联通用户请查看[设备接入服务](#)。

固件（Firmware）一般是指设备硬件的底层“驱动程序”，承担着一个系统最基础最底层工作的软件，比如计算机主板上的基本输入/输出系统BIOS（Basic Input/output System）。固件升级又称为FOTA（Firmware Over The Air），是指用户可以通过OTA的方式对支持LWM2M协议的设备进行固件升级，固件升级协议为LWM2M协议。

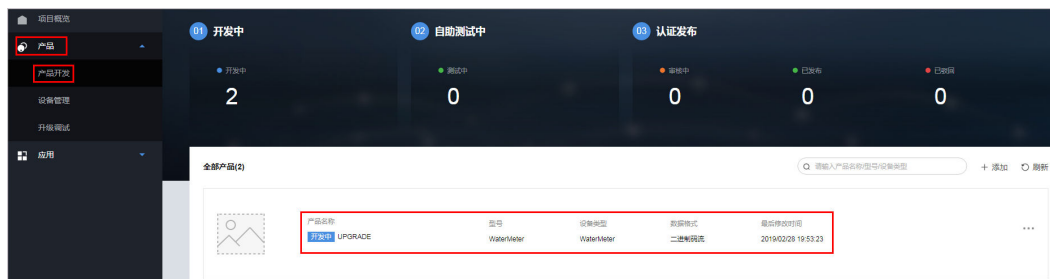
本章节将介绍基于开发中心的固件升级将如何进行调测。基于设备管理服务的固件升级可参考[设备管理服务固件升级](#)。

### 检查设备的固件升级能力

在执行固件升级操作前，请确认该设备支持固件升级能力。

**步骤1** 登录开发中心，并进入到对应的项目。

**步骤2** 选择“产品 > 产品开发”，选择具体产品，进入该产品的开发空间。



**步骤3** 在“Profile定义 > OM维护”下，确认“固件升级”功能已开启。



----结束

### 上传固件包

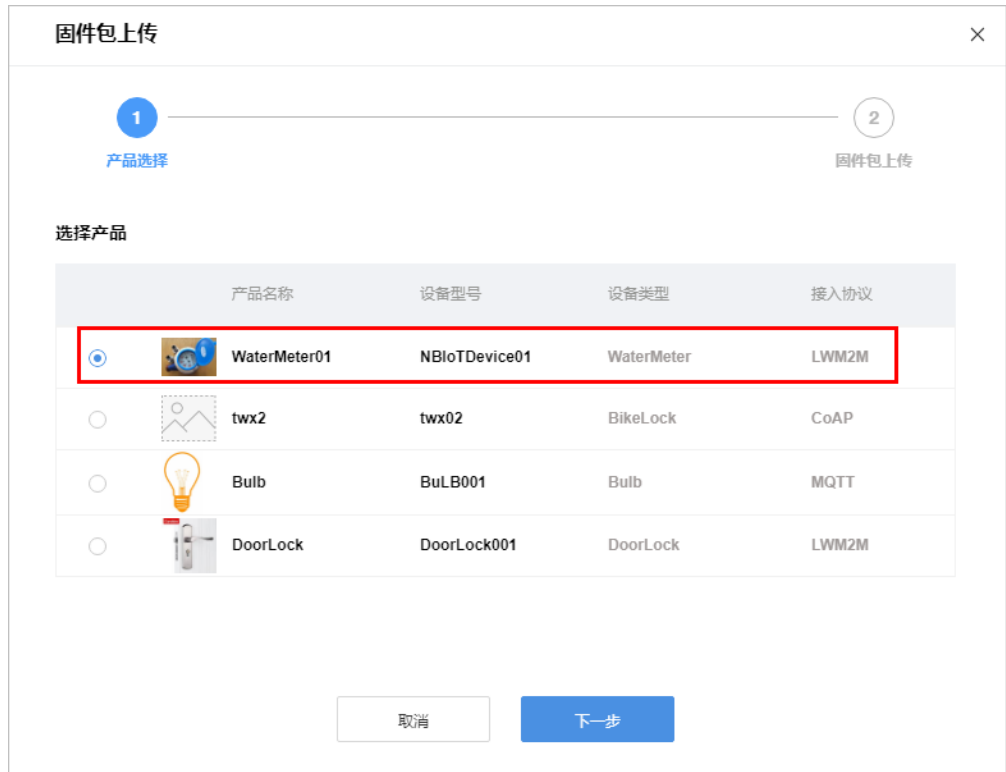
**步骤1** 选择“产品 > 升级调试 > 升级包管理 > 固件包管理”，点击“上传未签名固件包”。



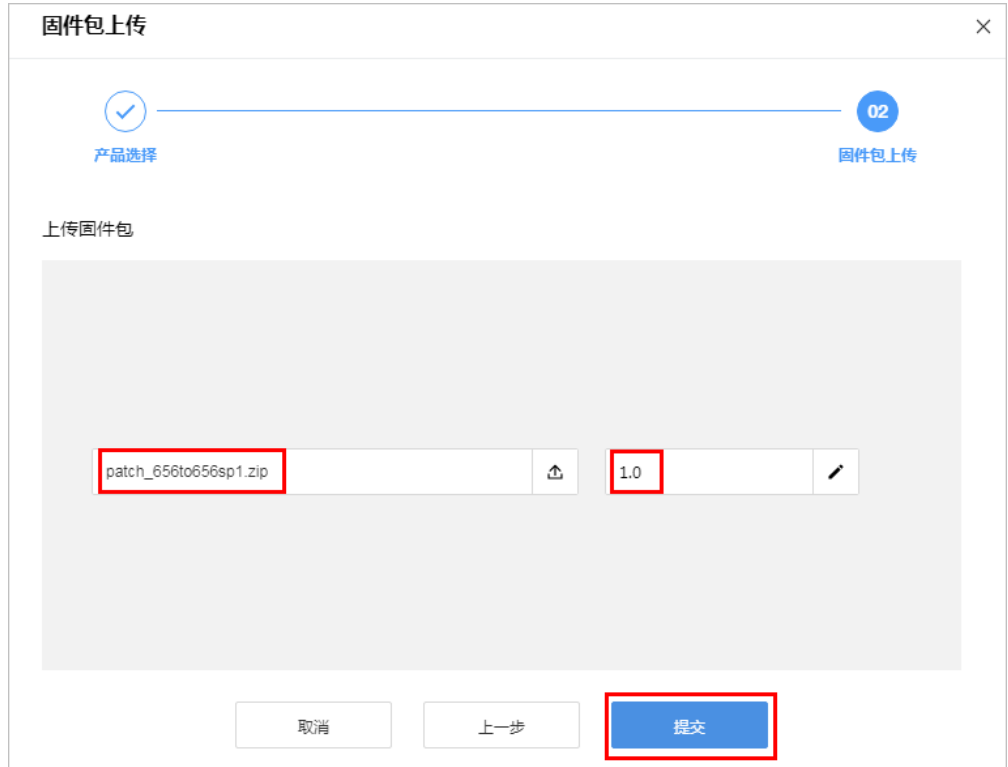
**步骤2** 根据固件包上传向导，完成固件包上传。

1. 选择产品，点击“下一步”。

因为固件包中只包含用于固件升级的bin文件，开发中心无法从固件包直接获取产品模型信息，所以此处需要选择产品并与固件包进行关联。



2. 选择需要上传的未签名固件包，填写固件包的版本号，点击“提交”。



----结束

## 创建固件升级任务

**步骤1** 选择“产品 > 升级调试 > 固件升级”，点击“创建升级任务”。



**步骤2** 根据固件升级向导，逐步完成升级任务的创建。

1. 填写任务基本信息，点击“下一步”。

如果需要配置升级任务的执行类型和重试参数，“高级”选择“是”。

- 执行类型：开发中心在什么时候给设备下发升级任务，取值范围：现在、设备在线时或自定义时间段。
- 重试参数：执行任务失败后，是否重新执行任务。

### 固件升级

01 任务信息      02 产品选择      03 固件包选择      04 设备选择

创建任务信息

\*任务名

高级

选择高级参数

执行类型

重试参数

\*重试次数





\*重试间隔（秒）

2. 选择产品，点击“下一步”。

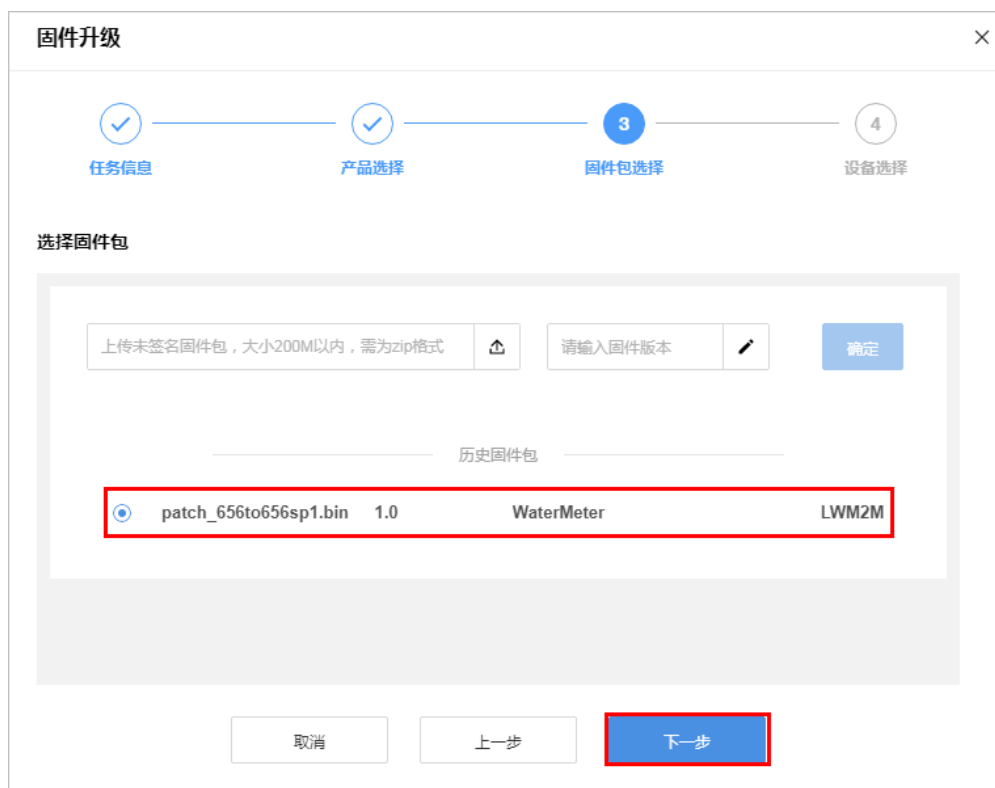
### 固件升级

1 任务信息      2 产品选择      3 固件包选择      4 设备选择

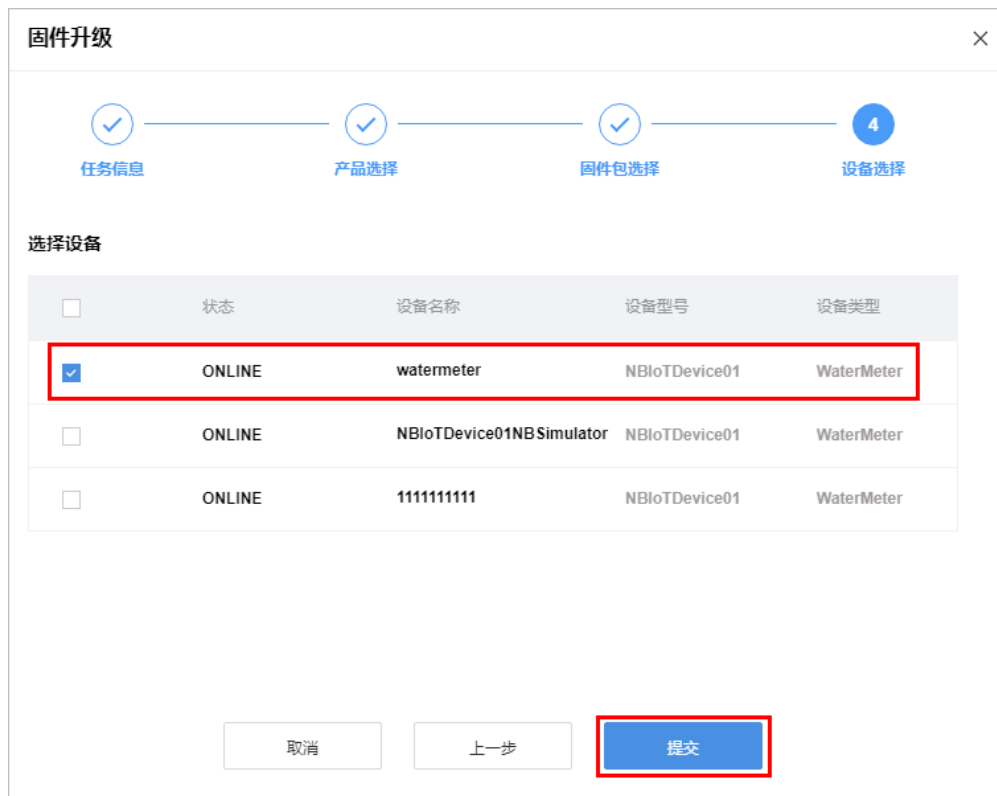
选择产品

	产品名称	设备型号	设备类型	接入协议
<input checked="" type="radio"/>	 WaterMeter01	NBLoTDevice01	WaterMeter	LWM2M
<input type="radio"/>	 twx2	twx02	BikeLock	CoAP
<input type="radio"/>	 Bulb	BuLB001	Bulb	MQTT
<input type="radio"/>	 DoorLock	DoorLock001	DoorLock	LWM2M

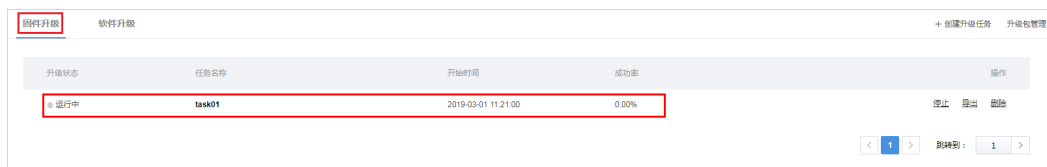
- 选择需要升级的固件包，点击“下一步”。  
如果“历史固件包”列表中没有需要的固件包，可以在此界面上上传新的未签名固件包，请确保上传的固件包是用于已选产品的固件升级。



- 在“设备选择”界面，会呈现已选产品对应的所有设备，可以选择一个或多个设备进行升级，点击“提交”。



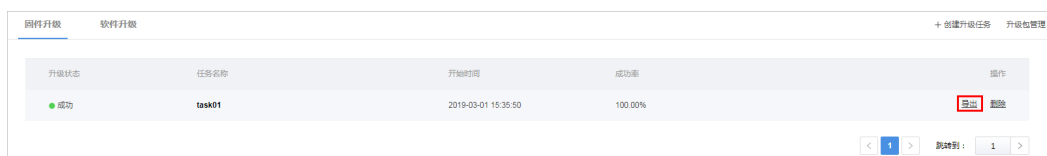
**步骤3** 固件升级任务创建后，可以在“固件升级”界面对固件升级任务进行管理。点击具体任务可以查看任务的“基本信息”和“升级详情”。



### 说明

设备在升级期间，设备不可进行业务交互。

**步骤4** 任务结束后，点击“导出”，导出任务详情文件。



----结束

## 4.6.1.6.2 软件升级（联通用户专用）

非联通用户请查看[设备接入服务](#)。

软件（Software）一般分为系统软件和应用软件，系统软件实现设备最基本的功能，比如编译工具、系统文件管理等；应用软件可以根据设备的特点，提供不同的功能，比如采集数据、数据分析处理等。软件升级又称为SOTA（SoftWare Over The Air），是指用户可以通过OTA的方式支持对LWM2M协议的设备进行软件升级，软件升级协议为PCP协议。

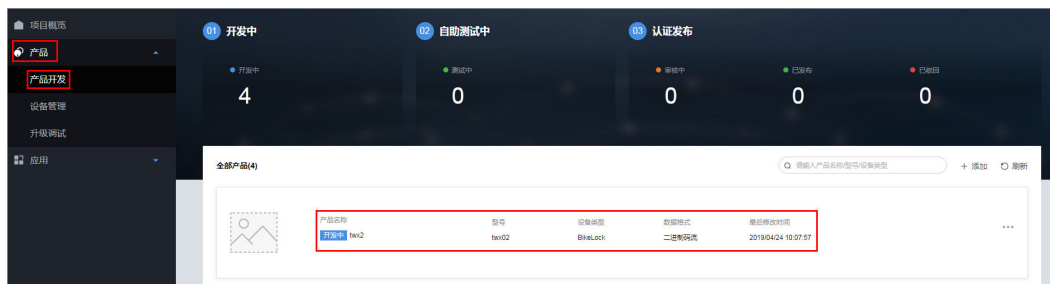
本章节将介绍基于开发中心的软件升级如何进行调测。基于设备管理服务的软件升级可参考[设备管理服务软件升级](#)。

## 检查设备软件升级能力

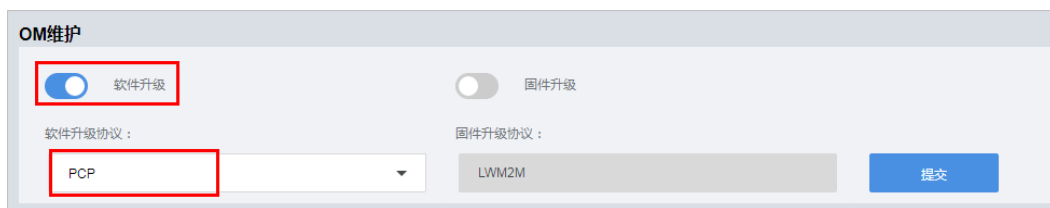
在执行软件升级操作前，请确认该设备支持软件升级能力。

**步骤1** 登录开发中心，并进入到对应的项目。

**步骤2** 选择“产品 > 产品开发”，选择具体产品，进入该产品的开发空间。



**步骤3** 在“Profile定义 > OM维护”下，确认“软件升级”功能已开启。



----结束

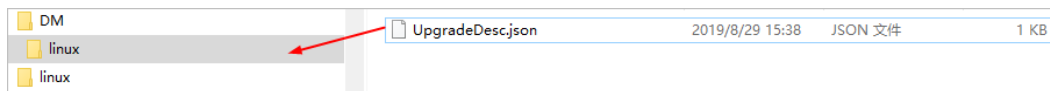
## 制作软件升级版本包

设备升级的软件包文件由各设备厂商提供，在物联网平台上传设备的软件升级包前，需要制作软件升级的版本包，用于修改软件包的描述文件，如软件版本、厂商名称、设备类型、产品模型等信息。下面将详细介绍版本包的制作方法。

**步骤1** 新建文件夹命名为“DM”，在DM文件夹下新建文件夹，命名为“linux”。

**步骤2** 使用Notepad++文本工具新建一个文本文件，拷贝如下内容到文本中，在Notepad++工具的“编码”菜单中选择“使用UTF-8编码”，然后将文本进行存储，存储路径选择步骤1中的“linux”文件夹，文件名称命名为“UpgradeDesc”，保存类型选择“json”。

```
{
  "specVersion": "",
  "fileName": "",
  "packageType": "",
  "version": "",
  "deviceType": "",
  "manufacturerName": "",
  "model": "",
  "protocolType": "",
  "description": "",
  "versionCheckCode": "",
  "deviceShard": "",
  "platform": "",
  "supportSourceVersionList": [],
  "date": ""
}
```

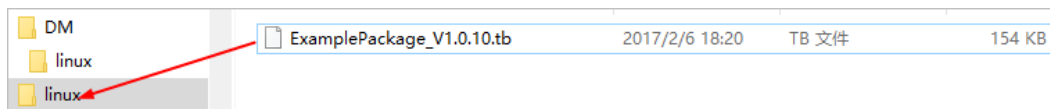


**步骤3** 打开创建的“UpgradeDesc.json”文件，修改软件升级描述文件，相关字段如下表所示。

字段名	字段描述	是否必填
specVersion	描述文件版本号，固定值：“1.0”。	是
fileName	软件包文件名，例如：“ExamplePackage_V1.0.10.xx”。	是
packageType	软件包类型，必须设置为：“softwarePackage”。	是
version	软件包版本号长度不超过16个字节，例如：“V1.0.10”	是
deviceType	设备类型，需要与产品模型保持一致。例如：“WaterMeter”。	是
manufacturerName	制造商名称，需要与产品模型保持一致。例如：“TestUtf8ManuName”。	是
model	产品型号，需要与产品模型保持一致。例如：“TestUtf8ModelM2M”。	是
protocolType	设备接入协议类型，需要与产品模型保持一致。例如：“CoAP”。 <b>说明</b> 设备接入的协议类型有三种：“CoAP”、“LWM2M”、“MQTT”。	是
description	对软件包的自定义描述。	是
versionCheckCode	软件升级包校验码，长度为4个字符。软件包下载时支持断点续传，根据该字段标识前后两次下载的软件包分片是否为同一个软件包。	否
deviceShard	终端下载软件包的每个分片的大小，单位为byte，如果不设置默认为500byte。大小为32~500之间。	否
platform	标识设备的操作系统，如linux。	否
supportSourceVersionList	支持用于升级此版本包的设备源版本列表。支持通配符配置，*代表匹配任意0~n个字符，?代表匹配单个任意字符，如果存在多个版本，请使用英文“;”隔开。	否
date	出包时间，格式为：“yyyy-MM-dd”。	否

**步骤4** 在与“DM”同级目录下创建文件夹，命名为“linux”，该文件夹名称必须同**步骤1**中的文件夹命令保持一致，将厂商软件包（软件包格式无限制）置于该文件中。





**步骤5** 选中“DM”和“linux”文件夹，使用压缩工具打包成ZIP格式的压缩包，建议命令为“xx\_package.zip”。

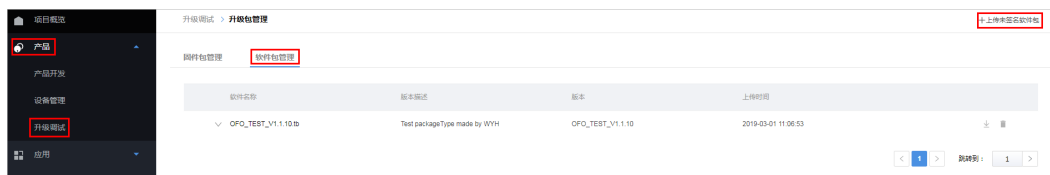
### 须知

- 文件“DM”和“linux”的命名是固定的。
- “xx\_package.zip”下不能包含package这层目录。
- 仅支持ZIP格式的压缩包，不能压缩成其他格式后，例如rar，再手动修改文件类型为zip。

----结束

## 上传软件包

**步骤1** 选择“产品 > 升级调试 > 升级包管理 > 软件包管理”，点击“上传未签名软件包”。



**步骤2** 在“软件包上传”界面，选择需要上传的未签名软件包，然后点击“提交”。

上传软件包前请确保开发中心已存在对应的产品模型。

### 说明

因为开发中心可以从软件包里的json文件获取产品模型信息，所以此处不需要像“固件包上传”一样，选择产品关联软件包。

----结束

## 创建软件升级任务

**步骤1** 选择“产品 > 升级调试 > 软件升级”，点击“创建升级任务”。



**步骤2** 根据软件升级向导，逐步完成升级任务的创建。

1. 填写任务基本信息，点击“下一步”。

如果需要配置升级任务的执行类型和重试参数，“高级”选择“是”。

- 执行类型：开发中心在什么时候给设备下发升级任务，取值范围：现在、设备在线时或自定义时间段。

- 重试参数：执行任务失败后，是否重新执行任务。

### 软件升级

1 2 3  
任务信息 软件包选择 设备选择

**创建任务信息**

\*任务名

高级

**选择高级参数**

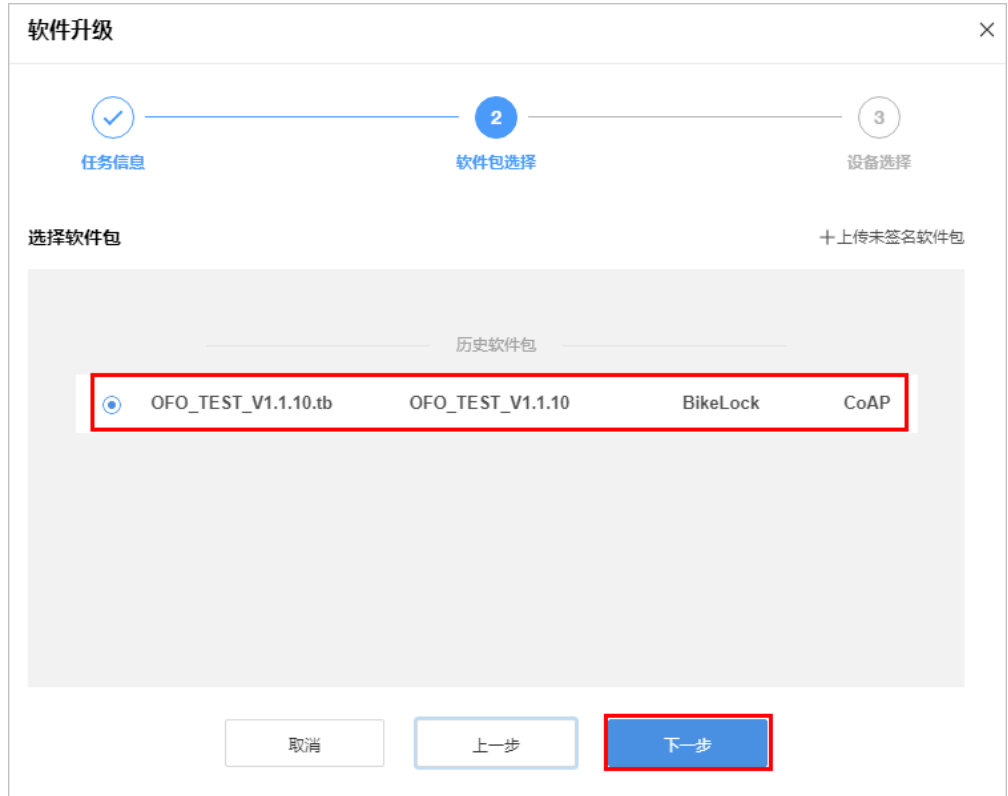
执行类型

重试参数

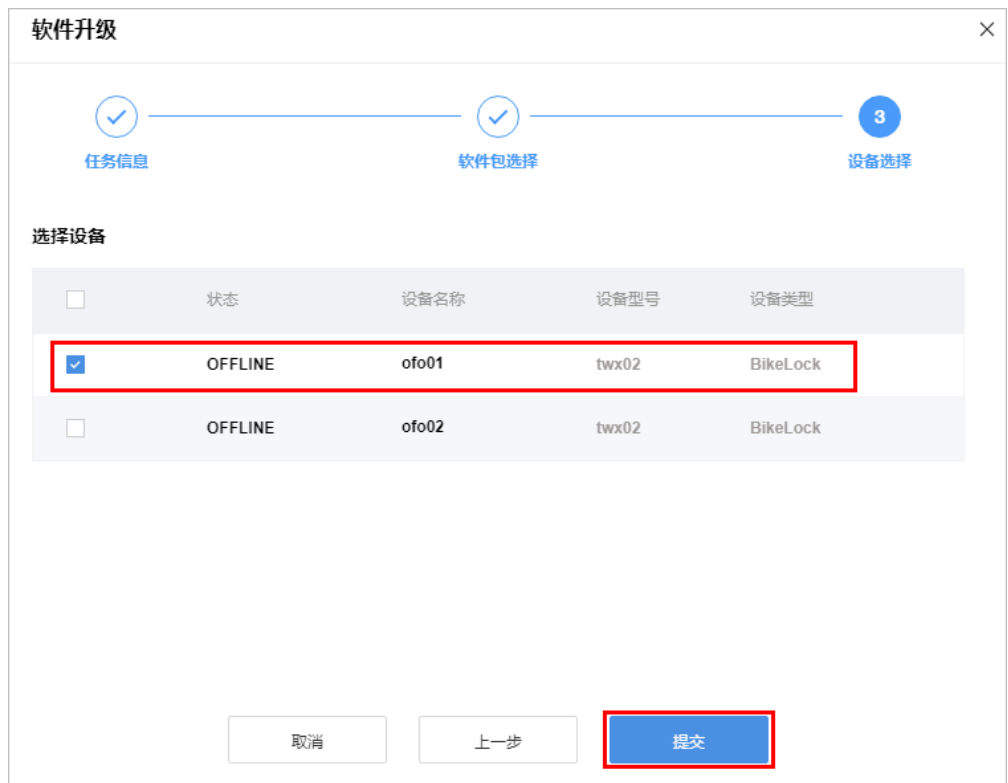
\*重试次数

\*重试间隔 (秒)

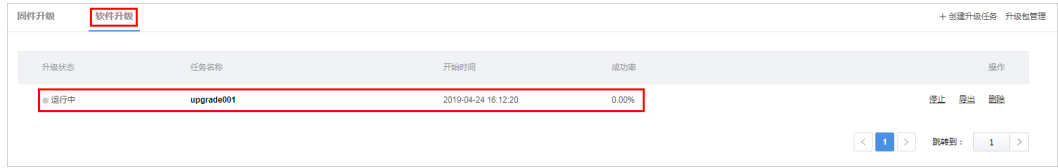
2. 选择需要升级的软件包，点击“下一步”。  
如果“历史软件包”列表中没有需要的软件包，可以点击“上传未签名软件包”上传新的未签名软件包。上传软件包前请确保开发中心已存在对应的产品模型。



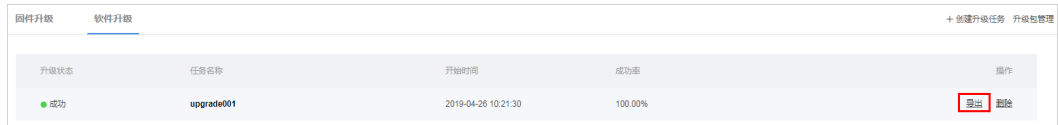
3. 在“设备选择”界面，可以选择一个或多个设备进行升级，点击“提交”。



**步骤3** 软件升级任务创建后，可以在“软件升级”界面对软件升级任务进行管理。点击具体任务可以查看任务的“基本信息”和“升级详情”。



**步骤4** 任务结束后，点击“导出”，导出任务详情文件。



----结束

## 4.6.1.7 应用侧开发

### 4.6.1.7.1 使用 API 对接

#### 4.6.1.7.1.1 API 使用指导（联通用户专用）

非联通用户请查看[设备接入服务](#)。

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Restful API（Application Programming Interface）。您可以调用开放的API，快速集成物联网平台的功能，如设备管理、数据采集、命令下发和消息推送等功能。

## 接口介绍

API分组	应用场景
<a href="#">应用安全接入</a>	<p>本部分接口提供“鉴权”以及“刷新token”两个接口。应用服务器通过调用这两个接口获取鉴权令牌，是调用其他平台API的前提，其他接口调用都需要在请求的Header中携带参数app_key和Authorization。</p> <ul style="list-style-type: none"> <li>app_key取值与请求参数中appId相同。</li> <li>Authorization的格式为Authorization: Bearer {accessToken}。</li> </ul> <p>关于应用安全接入的接口一般有两种调用模式，一种是每次业务处理前都调用一下鉴权接口重新获取token，然后用token来调用其他业务接口；另外一种应用服务器有一个有效管理机制，通过定期地刷新token，保证在调用其他业务接口前token都是有效的。</p>

API分组	应用场景
<a href="#">设备管理</a>	<p>本部分接口提供设备管理的相关功能。设备作为一种资源，设备管理接口提供了注册设备，删除设备，修改设备信息、位置信息等。查询设备信息的接口归类在“数据采集”中。</p> <p>平台的接入的设备有两种，直连设备和非直连设备。</p> <ul style="list-style-type: none"> <li>直连设备指已实现TCP/IP协议栈的设备，它可以直接与平台进行通信，常见的设备包括网关、以太网设备、NB-IoT设备等。</li> <li>非直连设备指未实现TCP/IP协议栈的设备，只能基于一些简单的近场通信协议如ZigBee、ZWave、Bluetooth或者是其他的一些非IP有线方式传输协议如串口、并口等接入，此时需要一个网关设备，先让设备接入到网关，再通过网关与华为物联网平台通信。</li> </ul> <p>设备管理的接口中提供了发现非直连设备、删除非直连设备等接口。如果您的方案中不包含非直连设备，则不需要调用这两个接口。</p>
<a href="#">数据采集</a>	<p>数据采集接口提供设备的查询、批量设备查询、设备历史数据和历史命令的查询、设备能力（即profile中定义的设备能力）查询。</p> <p>当应用服务器向平台查询设备的历史数据和历史命令时，这些数据已经存储在物联网平台上，因此与设备是否在线无关。</p>
<a href="#">订阅管理</a>	<p>物联网平台允许应用服务器订阅其所关注的事件，每一种事件的每一次订阅都会生成一个<b>subscriptionId</b>，应用服务器可以使用<b>subscriptionId</b>对本次订阅进行查询、更新、删除等操作。</p> <p>平台是通过Restful接口向应用服务器推送数据的，因此应用服务器需要开发一个Restful接口来接收推送数据，这个接口的URL也需要在订阅时提供给物联网平台。由于不同事件的推送消息结构不同，因此应用服务器可以考虑为每一种事件实现一个Restful接口。</p> <p><b>注意事项：</b></p> <ul style="list-style-type: none"> <li>关于平台上的事件种类和推送的消息样例，请参考<a href="#">推送通知</a>。</li> <li>https推送的前提是先在平台上上传应用服务器的<b>CA证书</b>。</li> </ul>
<a href="#">命令下发</a>	<p>本部分接口提供命令的创建、删除、查询、批量创建等功能。</p>
<a href="#">批量处理</a>	<p>目前仅支持创建批量下发缓存命令任务，您也可以查询批量任务信息。</p>
<a href="#">设备组管理</a>	<p>应用服务器可以通过本部分接口在物联网平台上管理设备组。设备组是把设备进行分组管理。一个设备可以归属到多个设备组内。</p> <p>在对设备进行某些操作时（如升级设备软固件、批量下发命令等），可通过设备组来指定要进行操作的设备。</p>
<a href="#">设备升级</a>	<p>若需要对设备进行软固件版本升级，应用服务器可调用本部分接口为多个设备创建升级任务，升级前请确保目标版本包已经上传到物联网平台。当前仅支持对CoAP接入的设备进行软固件版本升级。</p>

#### 4.6.1.7.1.2 使用 Postman 调测（联通用户专用）

非联通用户请查看[设备接入服务](#)。

调用API接口前建议完成Profile文件和编解码插件的开发。尽管接口的调用不依赖于Profile文件和编解码插件，但是由于profile文件定义了设备数据的字段，编解码插件

是上报数据和下发命令时的必要条件，因此为了使得调用接口前的业务都正常，需要先行完成产品的开发。


为充分了解接口，建议提前获取《[应用侧API参考](#)》查阅。我们已经写好了Postman的 **Collection**，在Collection中接口的请求结构体已经完成可以直接使用。

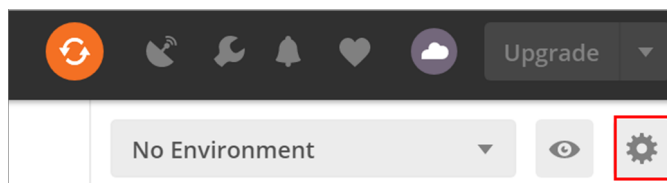
本文档以Postman为例，模拟应用服务器以HTTPS协议接入物联网平台，调测以下API接口：

- “[鉴权](#)”接口
- “[注册设备](#)”接口
- “[修改设备信息](#)”接口
- “[删除设备](#)”接口

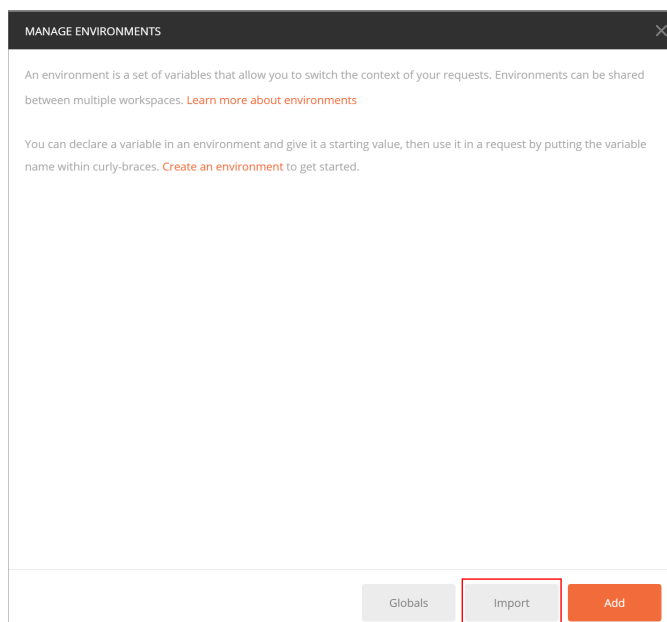
## 配置 Postman

**步骤1** 导入Postman环境变量。

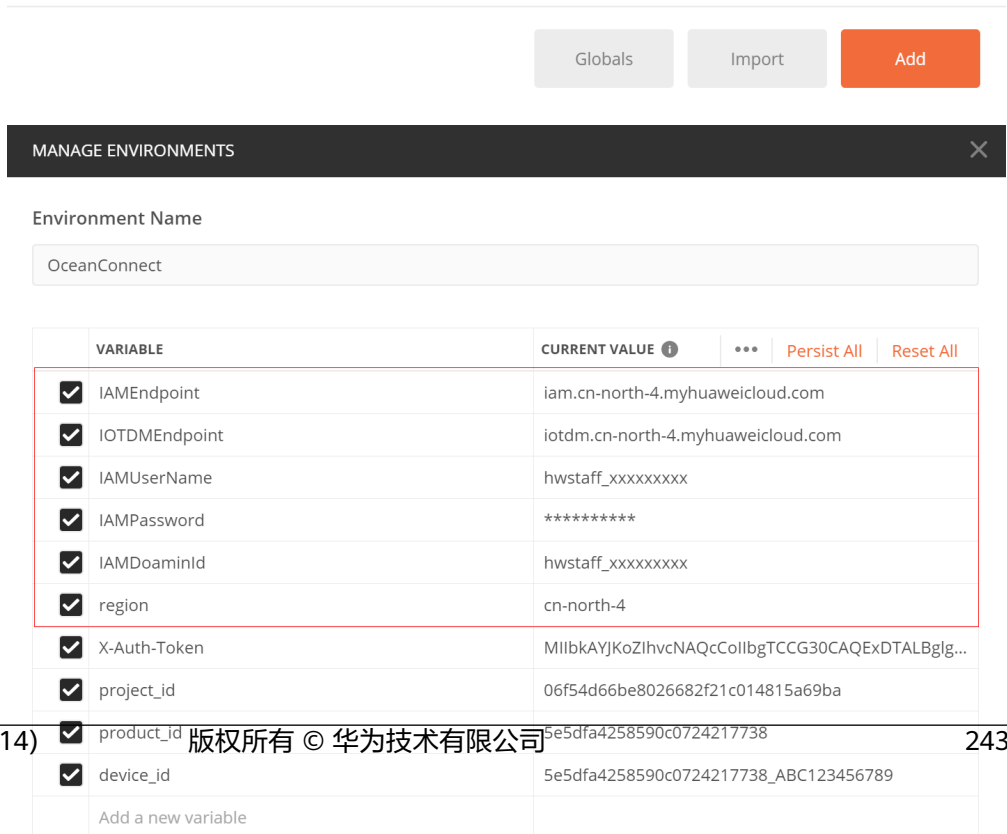
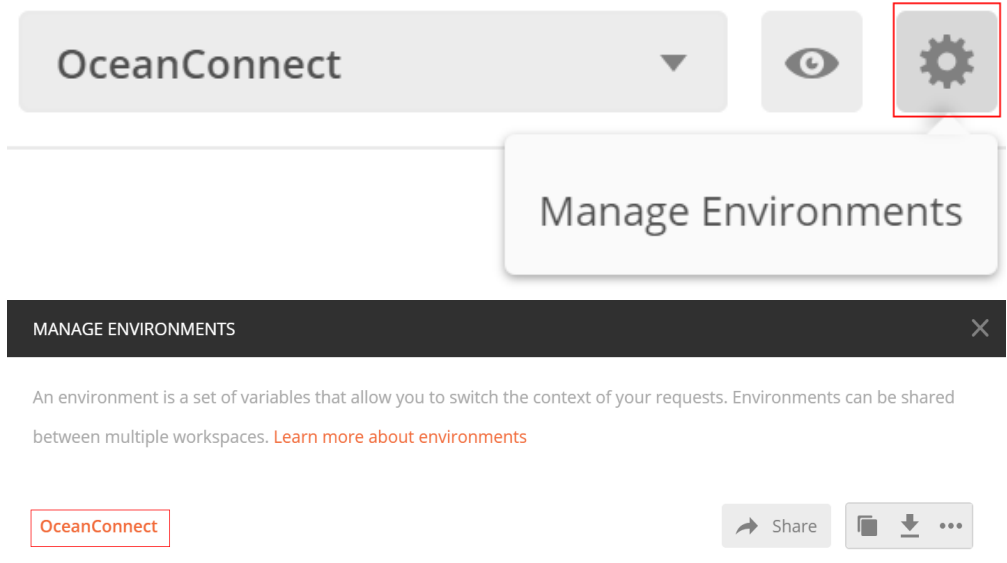
1. 点击右上角的  图标，打开“MANAGE ENVIRONMENTS”窗口。



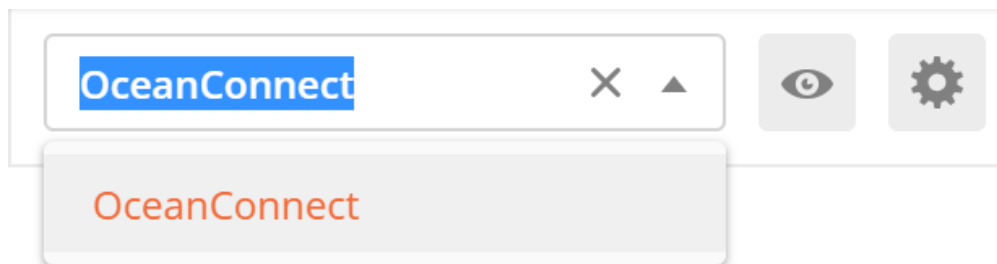
2. 点击“Import”，导入OceanConnect.postman\_environment.json文件。



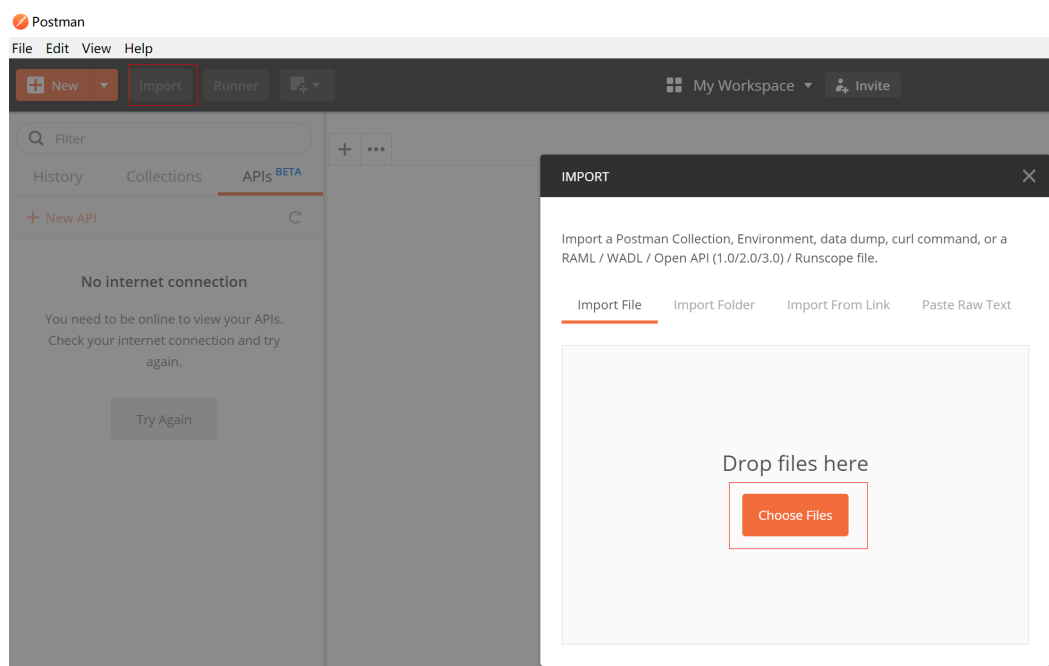
3. 点击“**Manage Environments**”，选择导入的“**OceanConnect**”环境，修改IAM终端节点“**IAMEndpoint**”、物联网平台终端节点“**IOTDMEndpoint**”、华为云用户名“**IAMUserName**”、华为云密码“**IAMPassword**”、华为云账号名“**IAMDoaminId**”、区域“**region**”。这里已经填写好北京四站点对应的IAM终端节点、物联网平台终端节点和区域，如果您是在北京四站点订购的“设备管理”服务，只需要修改华为云用户名、华为云密码和华为云账号名。



4. 返回主页，选择环境变量为刚导入的“OceanConnect”。



- 步骤2 导入“应用侧API调用（V5版本）.postman\_collection.json”。



成功后：





----结束

## 调测“获取 IAM 用户 Token”接口

在访问物联网平台业务接口前，应用服务器需要调用“获取IAM用户Token”接口鉴权，华为云认证通过后向应用服务器返回鉴权令牌X-Subject-Token。

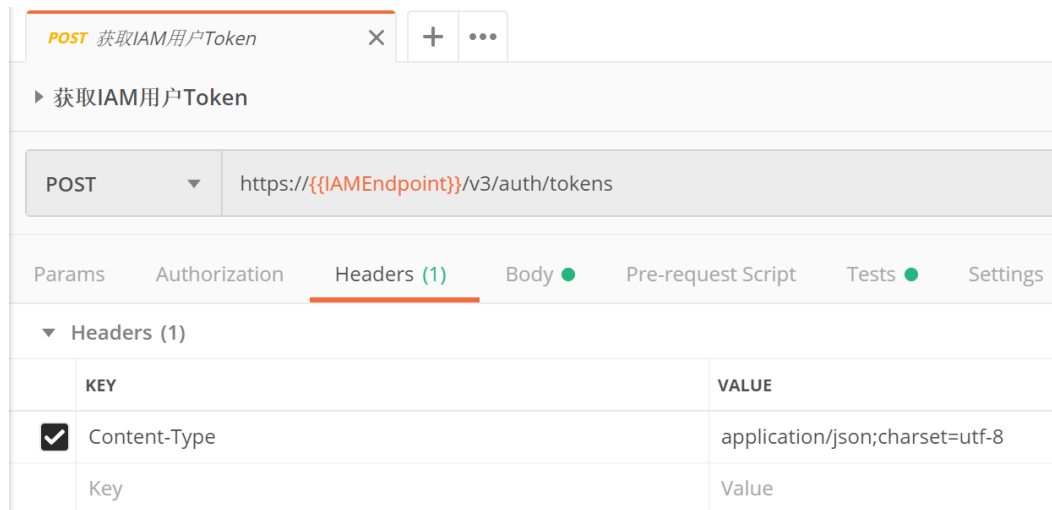
应用服务器需要构造一个HTTP请求，请求示例如下：

```
POST https://iam.cn-north-4.myhuaweicloud.com/v3/auth/tokens
Content-Type: application/json
```

```
{
  "auth": {
    "identity": {
      "methods": [
        "password"
      ],
      "password": {
        "user": {
          "name": "username",
          "password": "*****",
          "domain": {
            "name": "domainname"
          }
        }
      }
    },
    "scope": {
      "project": {
        "name": "xxxxxxxx"
      }
    }
  }
}
```

接下来参考API文档，调测“获取IAM用户Token”接口。

### 步骤1 配置“获取IAM用户Token”接口的HTTP方法、URL和Headers。

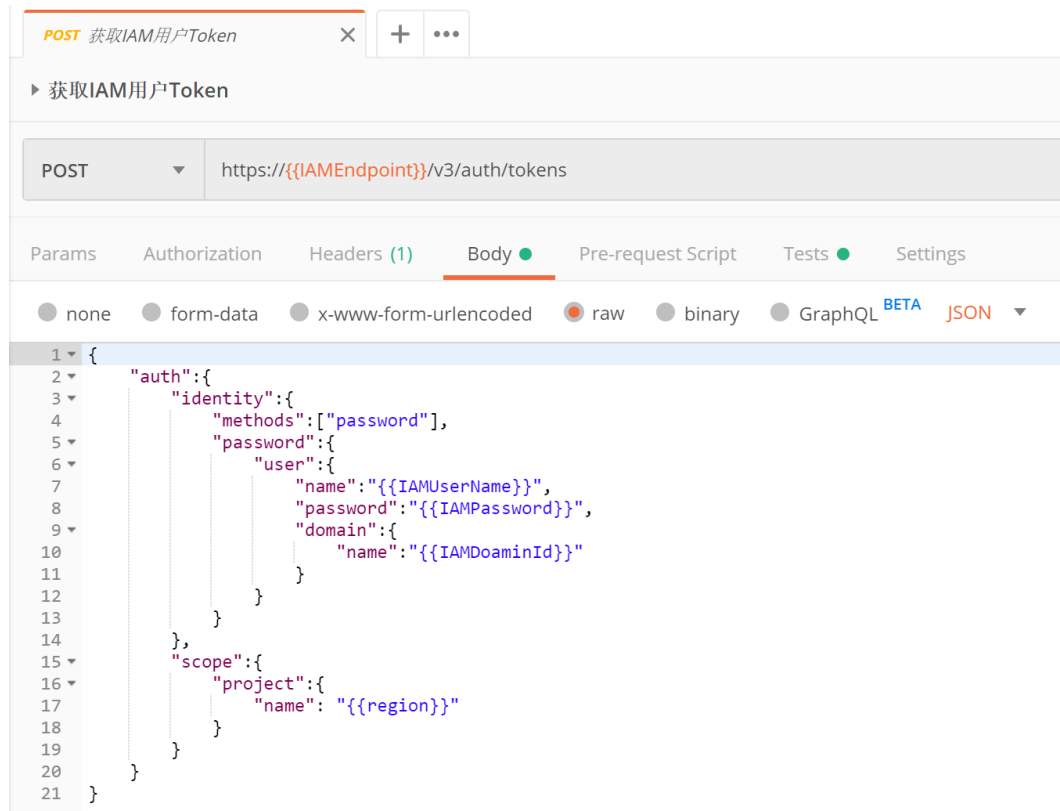


The screenshot shows an API client interface with the following configuration:

- Method: POST
- URL: https://{{IAMEndpoint}}/v3/auth/tokens
- Headers (1):

KEY	VALUE
<input checked="" type="checkbox"/> Content-Type	application/json; charset=utf-8
Key	Value

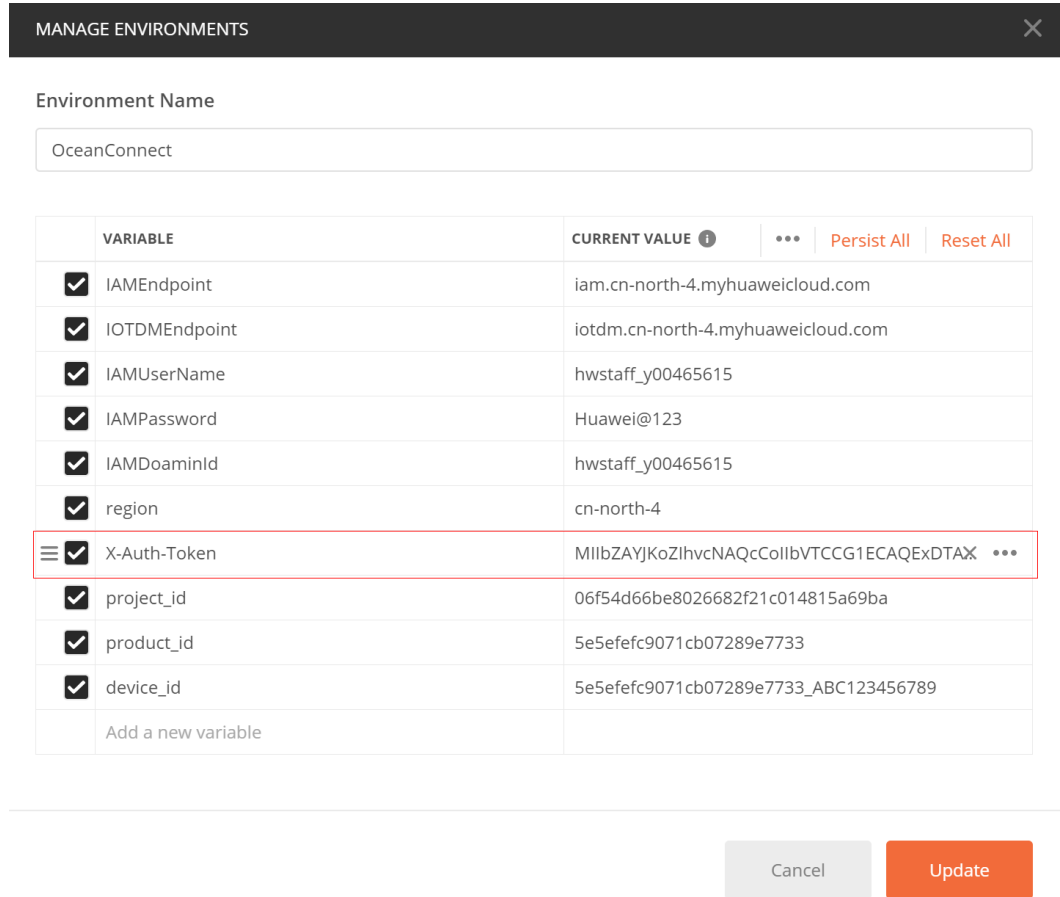
### 步骤2 配置“获取IAM用户Token”接口的Body。



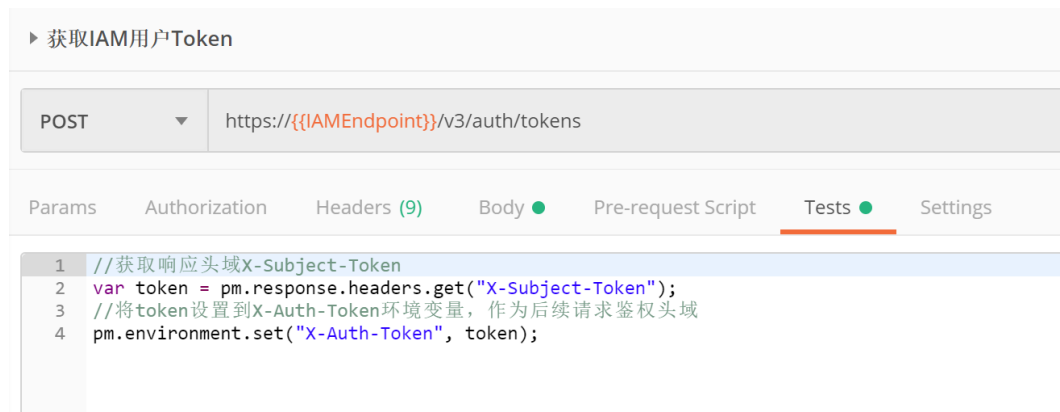
**步骤3** 点击“Send”，在下方查看返回码和响应消息内容。

KEY	VALUE
Date	Wed, 04 Mar 2020 01:00:53 GMT
Content-Type	application/json; charset=UTF-8
Content-Length	18468
Connection	keep-alive
X-IAM-Trace-Id	token_cn-north-4_null_5e627fb3ddfc776374456e059c3666a8
Cache-Control	no-cache, no-store, must-revalidate
Pragma	no-cache
Expires	Thu, 01 Jan 1970 00:00:00 GMT
X-Subject-Token	MllbZAYJKoZlhcNAQcCollbVTCGG1ECAQExDTALBglghkgBZQMEAgEwghI2BgkqhkiG9w0BBwGggh...
X-Request-Id	c93c1b0311803c589f61b89ef900b48d
Server	api-gateway
Strict-Transport-Security	max-age=31536000; includeSubdomains;
X-Frame-Options	SAMEORIGIN
X-Content-Type-Options	nosniff
X-Download-Options	noopen
X-XSS-Protection	1; mode=block;

**步骤4** 请将返回头域中的X-Subject-Token更新到“OcceanConnect”环境的“X-Auth-Token”参数中，以便于在调用其它接口时使用。若超过令牌有效时间，需要重新调用鉴权接口。



这里我们已经在postman中自动更新了“X-Auth-Token”参数，使用时无需手动操作。



----结束

## 调测“查询IAM用户可以访问的项目列表”接口

在访问物联网平台业务接口前，应用服务器需要调用“查询IAM用户可以访问的项目列表”接口获取用户的项目ID，用于后续访问物联网平台业务接口。

应用服务器需要构造一个HTTP请求，请求示例如下：

```
GET https://iam.cn-north-4.myhuaweicloud.com/v3/auth/projects
Content-Type: application/json
X-Auth-Token: *****
```

接下来参考API文档，调测“查询IAM用户可以访问的项目列表”接口。

### 步骤1 配置“查询IAM用户可以访问的项目列表”接口的HTTP方法、URL和Headers。

▶ 查询IAM用户可以访问的项目列表

GET <https://{{IAMEndpoint}}/v3/auth/projects>

Params Authorization Headers (9) Body Pre-request Script Tests ● Settings

▼ Headers (2)

KEY	VALUE
<input checked="" type="checkbox"/> Content-Type	application/json
<input checked="" type="checkbox"/> X-Auth-Token	{{X-Auth-Token}}

### 步骤2 点击“Send”，在下方查看返回码和响应消息内容。

Body Cookies Headers (15) Test Results Status: 200 OK

Pretty Raw Preview Visualize BETA JSON ↕

```
1 {
2   "projects": [
3     {
4       "domain_id": "ba21fb12cfc440569954a2ac9a99323a",
5       "is_domain": false,
6       "parent_id": "ba21fb12cfc440569954a2ac9a99323a",
7       "name": "ap-southeast-1",
8       "description": "",
9       "links": {
10        "self": "https://iam.myhuaweicloud.com/v3/projects/072a8dbc980100d2f0ec0146f237196"
11      },
12      "id": "072a8dbc980100d2f0ec0146f237196",
13      "enabled": true
14    },
15    {
16      "domain_id": "ba21fb12cfc440569954a2ac9a99323a",
17      "is_domain": false,
18      "parent_id": "ba21fb12cfc440569954a2ac9a99323a",
19      "name": "MOS",
20      "description": "",
21      "links": {
22        "self": "https://iam.myhuaweicloud.com/v3/projects/b6c7508ff62e4beb91cee1c1ce49ecd9"
23      },
24      "id": "b6c7508ff62e4beb91cee1c1ce49ecd9",
25      "enabled": true
26    }
27  ],
28 }
```

### 步骤3 返回body中包含一个projects列表，查找其中“name”参数值与“OceanConnect”环境中“region”参数值相同的条目，取其“id”参数值更新到“OceanConnect”环境中“project\_id”参数，以便于在调用其它接口时使用。

```

Body Cookies Headers (15) Test Results Status: 200 OK
Pretty Raw Preview Visualize BETA JSON
95     },
96     "id": "072a8dcdbd08026542f00c014ee62ff50",
97     "enabled": true
98   },
99   {
100     "domain_id": "ba21fb12cfc440569954a2ac9a99323a",
101     "is_domain": false,
102     "parent_id": "ba21fb12cfc440569954a2ac9a99323a",
103     "name": "cn-north-4",
104     "description": "",
105     "links": {
106       "self": "https://iam.myhuaweicloud.com/v3/projects/06f54d66be8026682f21c014815a69ba"
107     },
108     "id": "06f54d66be8026682f21c014815a69ba",
109     "enabled": true
110   },
111   {
112     "domain_id": "ba21fb12cfc440569954a2ac9a99323a",
113     "is_domain": false,
114     "parent_id": "ba21fb12cfc440569954a2ac9a99323a",
115     "name": "ap-southeast-3",
116     "description": "",
117     "links": {
118       "self": "https://iam.myhuaweicloud.com/v3/projects/072a8dcdbd08026502fb1c014ead6fc7a"
119     },
120     "id": "072a8dcdbd08026502fb1c014ead6fc7a",
121     "enabled": true
122   },
  
```

MANAGE ENVIRONMENTS
✕

Environment Name

OceanConnect

	VARIABLE	CURRENT VALUE ⓘ	⋮	Persist All	Reset All
<input checked="" type="checkbox"/>	IAMEndpoint	iam.cn-north-4.myhuaweicloud.com			
<input checked="" type="checkbox"/>	IOTDMEndpoint	iotdm.cn-north-4.myhuaweicloud.com			
<input checked="" type="checkbox"/>	IAMUserName	hwstaff_y00465615			
<input checked="" type="checkbox"/>	IAMPASSWORD	Huawei@123			
<input checked="" type="checkbox"/>	IAMDoaminId	hwstaff_y00465615			
<input checked="" type="checkbox"/>	region	cn-north-4	✕ ⋮		
<input checked="" type="checkbox"/>	X-Auth-Token	MllbZAYJKoZlhvcNAQcCollbVTCCG1ECAQExDTALBglg...			
<input checked="" type="checkbox"/>	project_id	06f54d66be8026682f21c014815a69ba			
<input checked="" type="checkbox"/>	product_id	5e5efefc9071cb07289e7733			
<input checked="" type="checkbox"/>	device_id	5e5efefc9071cb07289e7733_ABC123456789			
	Add a new variable				

Cancel

Update

这里我们已经在postman中自动更新了“project\_id”参数，使用时无需手动操作。

▶ 查询IAM用户可以访问的项目列表

GET https://{{IAMEndpoint}}/v3/auth/projects

Params Authorization Headers (9) Body Pre-request Script Tests ● Settings

```

1 var region = pm.environment.get("region");
2 var jsonData = pm.response.json();
3 var projects = jsonData.projects;
4 for (i = 0; i < projects.length; i++) {
5     if (projects[i].name == region) {
6         pm.environment.set("project_id", projects[i].id);
7     }
8 }

```

---结束

## 调测“创建产品”接口

在设备接入物联网平台前，应用服务器需要调用此接口创建产品，后续注册设备时需要使用这里创建的产品。

应用服务器需要构造一个请求，请求示例如下：

POST https://iotdm.cn-north-4.myhuaweicloud.com/v5/iot/{project\_id}/products  
Content-Type: application/json  
X-Auth-Token: \*\*\*\*\*

```

{
  "name": "Thermometer",
  "device_type": "Thermometer",
  "protocol_type": "LWM2M",
  "data_format": "binary",
  "manufacturer_name": "ABC",
  "industry": "smartCity",
  "description": "this is a thermometer produced by Huawei",
  "service_capabilities": [ {
    "service_type": "temperature",
    "service_id": "temperature",
    "description": "temperature",
    "properties": [ {
      "unit": "centigrade",
      "min": "1",
      "method": "R",
      "max": "100",
      "data_type": "decimal",
      "description": "force",
      "step": 0.1,
      "enum_list": [ "string" ],
      "required": true,
      "property_name": "temperature",
      "max_length": 100
    } ],
    "commands": [ {
      "command_name": "reboot",
      "responses": [ {
        "response_name": "ACK",
        "paras": [ {
          "unit": "km/h",
          "min": "1",
          "max": "100",
          "para_name": "force",
          "data_type": "string",

```

```

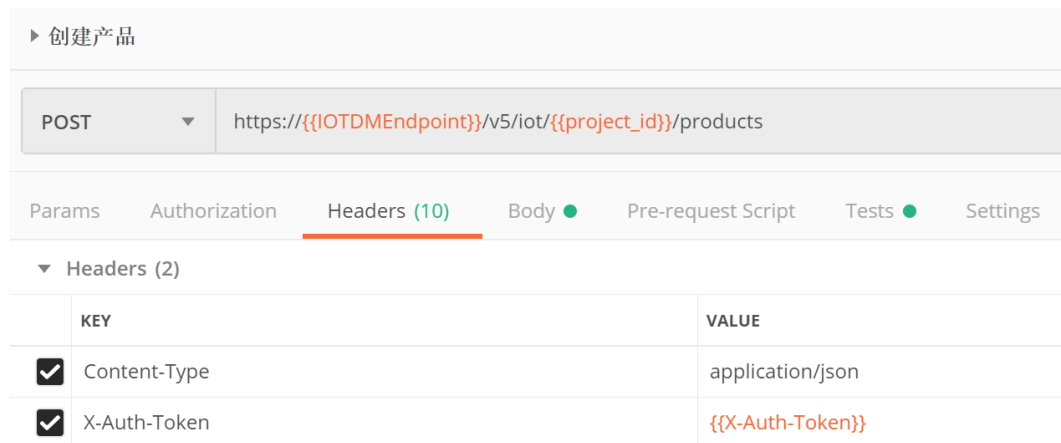
    "description": "force",
    "step": 0.1,
    "enum_list": [ "string" ],
    "required": false,
    "max_length": 100
  } ]
}],
"paras": [ {
  "unit": "km/h",
  "min": "1",
  "max": "100",
  "para_name": "force",
  "data_type": "string",
  "description": "force",
  "step": 0.1,
  "enum_list": [ "string" ],
  "required": false,
  "max_length": 100
} ]
}],
"option": "Mandatory"
}],
"app_id": "jeQDJQZltU8iKgFFoW060F5SGZka"
}

```

接下来参考API文档，调测物联网平台“创建产品”接口。

**注：**在以下步骤中，只呈现样例调测用到的参数。

### 步骤1 配置“创建产品”接口的HTTP方法、URL和Headers。



创建产品

POST ▼ https://{{IOTDMEndpoint}}/v5/iot/{{project\_id}}/products

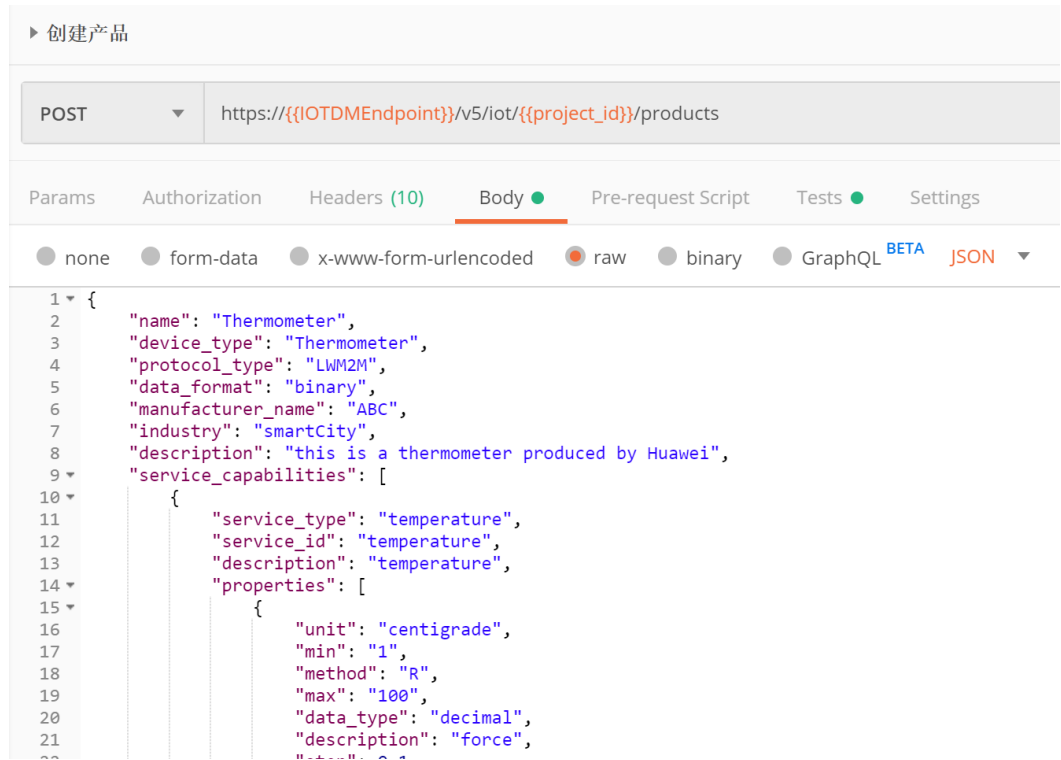
Params Authorization Headers (10) Body ● Pre-request Script Tests ● Settings

▼ Headers (2)

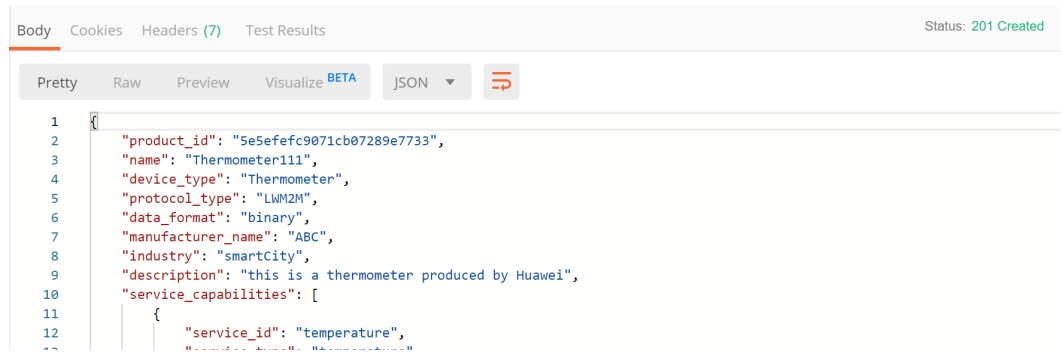
	KEY	VALUE
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	X-Auth-Token	{{X-Auth-Token}}

### 步骤2 配置“创建产品”接口的BODY。

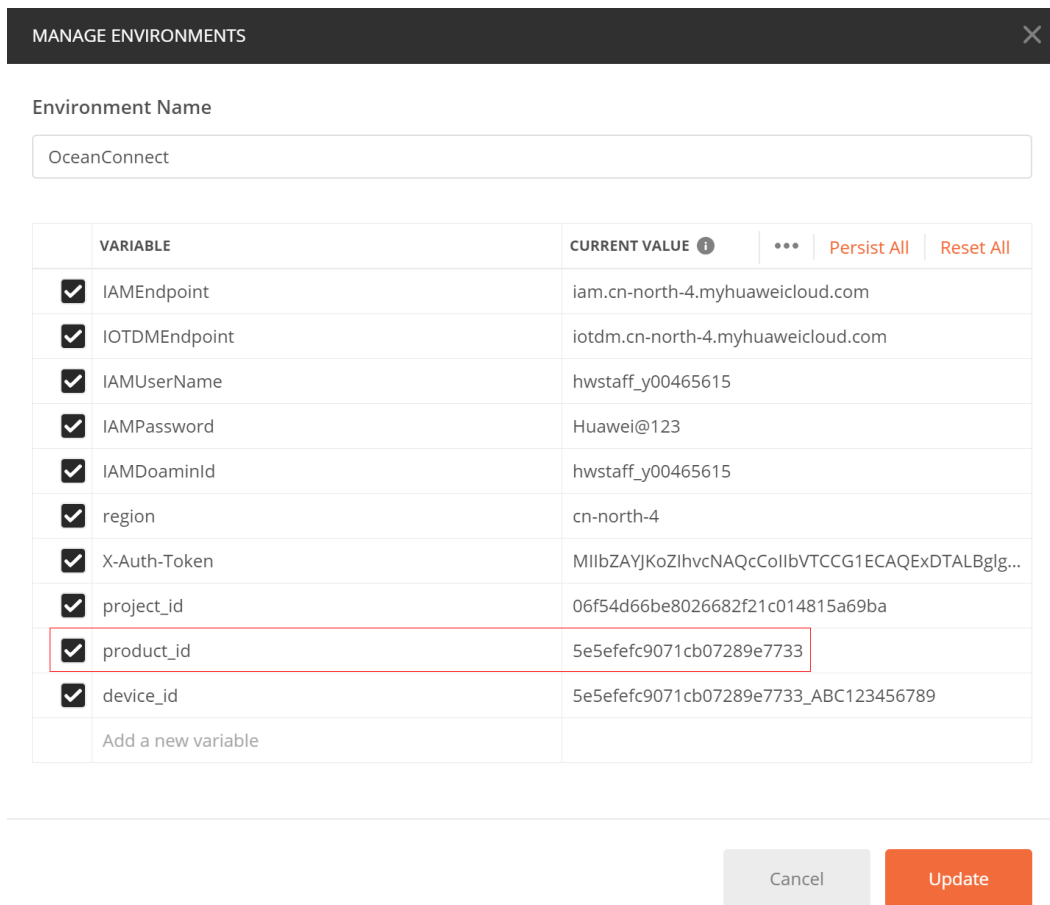




**步骤3** 点击“Send”，在下方查看返回码和响应消息内容。



**步骤4** 将返回的“product\_id”更新到“OceanConnect”环境中的“product\_id”参数中，用于后续其它接口使用。



这里我们已经在postman中自动更新了“**product\_id**”参数，使用时无需手动操作。



----结束

## 调测“查询产品”接口

应用服务器如果需要查询之前创建的产品详情，可以调用此接口。

应用服务器需要构造一个请求，请求示例如下：

```

GET https://iotdm.cn-north-4.myhuaweicloud.com/v5/iot/{project_id}/products/{product_id}
Content-Type: application/json
X-Auth-Token: *****
    
```

接下来参考API文档，调测物联网平台“查询产品”接口。

注：在以下步骤中，只呈现样例调测用到的参数。

### 步骤1 配置“查询产品”接口的HTTP方法、URL和Headers。

▶ 查询产品

GET `https://{{IOTDMEndpoint}}/v5/iot/{{project_id}}/products/{{product_id}}`

Params Authorization Headers (9) Body Pre-request Script Tests Settings

▼ Headers (2)

	KEY	VALUE
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	X-Auth-Token	{{X-Auth-Token}}

### 步骤2 点击“Send”，在下方查看返回码和响应消息内容。

Body Cookies Headers (14) Test Results Status: 200 OK

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "product_id": "5e5efefc9071cb07289e7733",
3   "name": "Thermometer111",
4   "device_type": "Thermometer",
5   "protocol_type": "LWM2M",
6   "data_format": null,
7   "manufacturer_name": "ABC",
8   "industry": "smartCity",
9   "description": "this is a thermometer produced by Huawei",
10  "service_capabilities": [
11    {
12      "service_id": "temperature",
13      "service_type": "temperature",
14      "properties": [
15        {
16          "property_name": null,
17          "required": true,
18          "data_type": null,
19          "enum_list": [],
20          "min": "1",
21          "max": "100",
22          "max_length": 1024,
23          "step": 0.1,
24          "unit": "centigrade",
25          "method": "R",
26          "description": null
27        }
28      ],
29      "commands": [
30    }
31  ]
32 }
```

----结束

## 调测“注册设备”接口

在设备接入物联网平台前，应用服务器需要调用此接口在物联网平台注册设备。在设备接入物联网平台时携带设备唯一标识，完成设备的接入认证。

应用服务器需要构造一个HTTP请求，请求示例如下：

```
POST https://iotdm.cn-north-4.myhuaweicloud.com/v5/iot/{project_id}/devices
Content-Type: application/json
X-Auth-Token: *****
{
```

```

"node_id": "ABC123456789",
"device_name": "dianadevice",
"product_id": "b640f4c203b7910fc3cbd446ed437cbd",
"auth_info": {
  "auth_type": "SECRET",
  "secure_access": true,
  "fingerprint": "dc0f1016f495157344ac5f1296335cff725ef22f",
  "secret": "3b935a250c50dc2c6d481d048cefdc3c",
  "timeout": 300
},
"description": "watermeter device"
}
    
```

接下来参考API文档，调测物联网平台“注册设备”接口。

**注：**在以下步骤中，只呈现样例调测用到的参数。

### 步骤1 配置“注册设备”接口的HTTP方法、URL和Headers。

注册设备

POST ▼ https://{{IOTDMEndpoint}}/v5/iot/{{project\_id}}/devices

Params Authorization Headers (10) Body ● Pre-request Script Tests ● Settings

▼ Headers (2)

	KEY	VALUE
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	X-Auth-Token	{{X-Auth-Token}}

### 步骤2 配置“注册设备”接口的Body。

注册设备

POST ▼ https://{{IOTDMEndpoint}}/v5/iot/{{project\_id}}/devices

Params Authorization Headers (10) Body ● Pre-request Script Tests ● Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL <sup>BETA</sup> JSON ▼

```

1 {
2   "node_id": "ABC123456789",
3   "device_name": "dianadevice",
4   "product_id": "{{product_id}}",
5   "auth_info": {
6     "auth_type": "SECRET",
7     "secure_access": true,
8     "fingerprint": "dc0f1016f495157344ac5f1296335cff725ef22f",
9     "secret": "3b935a250c50dc2c6d481d048cefdc3c",
10    "timeout": 300
11  },
12  "description": "watermeter device"
13 }
    
```

### 步骤3 点击“Send”，在下方查看返回码和响应消息内容。

```

1  {
2    "app_id": "PAutVGQZoEVJCncftia5MFeeU1Ea",
3    "device_id": "5e5efefc9071cb07289e7733_ABC123456789",
4    "node_id": "ABC123456789",
5    "gateway_id": "5e5efefc9071cb07289e7733_ABC123456789",
6    "device_name": "dianadevice",
7    "node_type": "GATEWAY",
8    "description": "watermeter device",
9    "fw_version": null,
10   "sw_version": null,
11   "auth_info": {
12     "auth_type": "SECRET",
13     "secret": "3b935a250c50dc2c6d481d048cefdc3c",
14     "fingerprint": null,
15     "secure_access": true,
16     "timeout": 300
17   },
18   "product_id": "5e5efefc9071cb07289e7733",
19   "status": "INACTIVE",
20   "create_time": "20200304T010621Z",
21   "tags": []
22 }

```

**步骤4** 请将返回的“**device\_id**”更新到“OceanConnect”环境中的“**device\_id**”参数中，用于后续其它接口使用。

MANAGE ENVIRONMENTS

Environment Name

OceanConnect

VARIABLE	CURRENT VALUE	...	Persist All	Reset All
<input checked="" type="checkbox"/> IAMEndpoint	iam.cn-north-4.myhuaweicloud.com			
<input checked="" type="checkbox"/> IOTDMEndpoint	iotdm.cn-north-4.myhuaweicloud.com			
<input checked="" type="checkbox"/> IAMUserName	hwstaff_y00465615			
<input checked="" type="checkbox"/> IAMPASSWORD	Huawei@123			
<input checked="" type="checkbox"/> IAMDoaminId	hwstaff_y00465615			
<input checked="" type="checkbox"/> region	cn-north-4			
<input checked="" type="checkbox"/> X-Auth-Token	MIIbZAYJKoZlhcNAQCcOllbVTCCG1ECAQEXDTALBgl...			
<input checked="" type="checkbox"/> project_id	06f54d66be8026682f21c014815a69ba			
<input checked="" type="checkbox"/> product_id	5e5efefc9071cb07289e7733			
<input checked="" type="checkbox"/> device_id	5e5efefc9071cb07289e7733_ABC123456789			
Add a new variable				

Cancel Update

这里我们已经在postman中自动更新了“**device\_id**”参数，使用时无需手动操作。



----结束

## 调测“查询设备”接口

应用服务器需要查询在物联网平台注册的设备详情时，可以调用此接口。

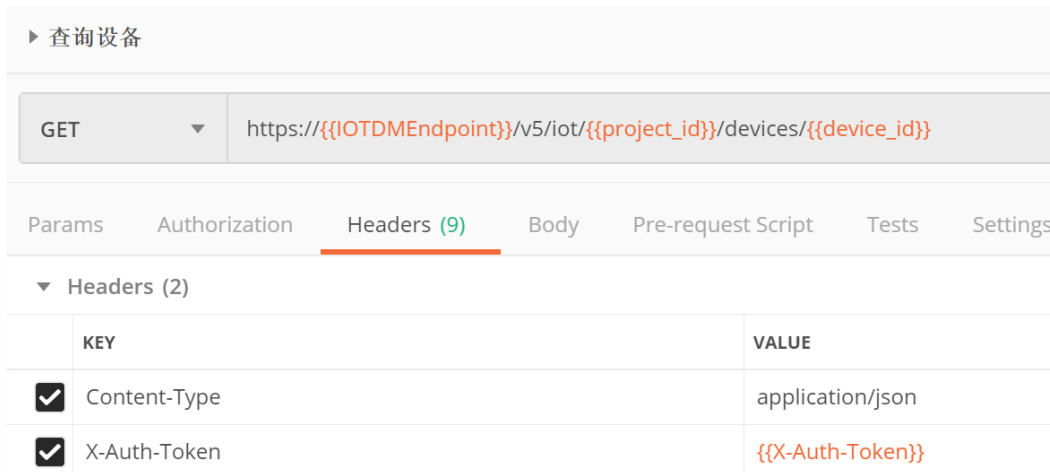
应用服务器需要构造一个HTTP请求，请求示例如下：

```
GET https://iotdm.cn-north-4.myhuaweicloud.com/v5/iot/{project_id}/devices/{device_id}
Content-Type: application/json
X-Auth-Token: *****
```

接下来参考API文档，调测物联网平台“查询设备”接口。

**注：**在以下步骤中，只呈现样例调测用到的参数。

### 步骤1 配置“查询设备”接口的HTTP方法、URL和Headers。



### 步骤2 点击“Send”，在下方查看返回码和响应消息内容。

```
Body Cookies Headers (14) Test Results Status: 200 OK
Pretty Raw Preview Visualize BETA JSON
1 {
2   "app_id": "PAutVGQZoEVJCncftia5MFeeU1Ea",
3   "device_id": "5e5efefc9071cb07289e7733_ABC123456789",
4   "node_id": "ABC123456789",
5   "gateway_id": "5e5efefc9071cb07289e7733_ABC123456789",
6   "device_name": "dianadevice",
7   "node_type": "GATEWAY",
8   "description": "watermeter device",
9   "fw_version": null,
10  "sw_version": null,
11  "auth_info": {
12    "auth_type": "SECRET",
13    "secret": "*****",
14    "fingerprint": null,
15    "secure_access": true,
16    "timeout": 0
17  },
18  "product_id": "5e5efefc9071cb07289e7733",
19  "status": "INACTIVE",
20  "create_time": "20200304T010621Z",
21  "tags": []
22 }
```

----结束

#### 4.6.1.7.1.3 使用 Java API Demo 调测（联通用户专用）

本文档基于调用API接口的代码样例（Java）进行指导，接口信息详见API参考文档。

### 准备 Java 开发环境

本小节以Windows操作系统为例，介绍安装JDK1.8以及安装Eclipse的方法，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1** 进入[Java JDK官网](#)，下载JDK1.8版本（比如：jdk-8u161-windows-x64.exe），双击进行安装。

**步骤2** 配置Java环境变量。

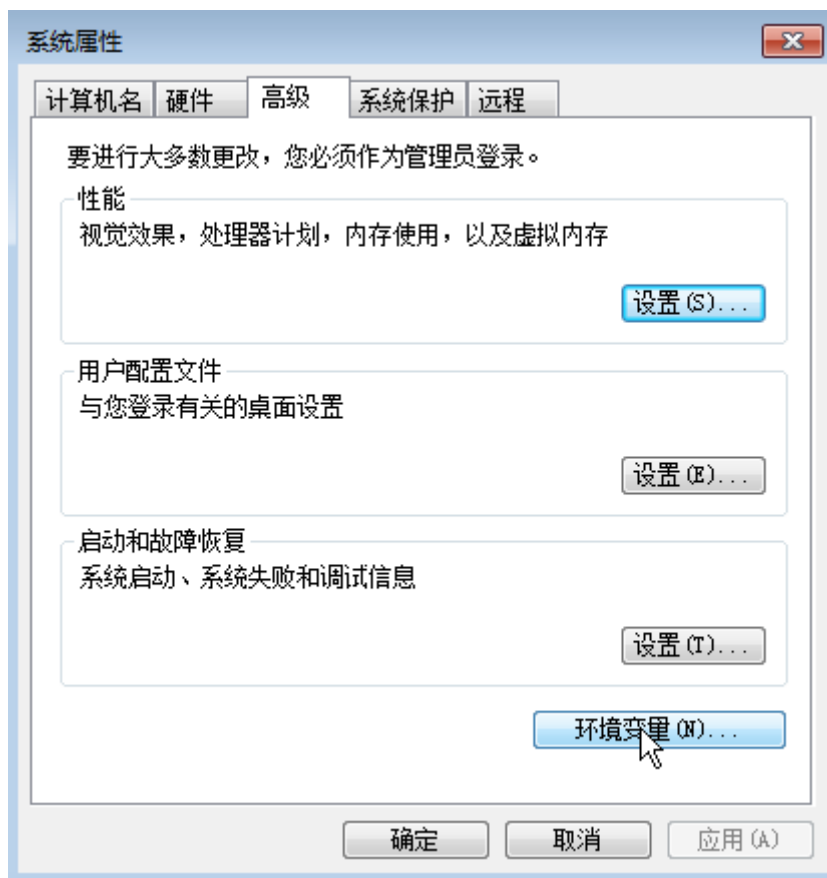
1. 右键单击“计算机”，选择“属性”。



2. 点击“高级系统设置”。



3. 点击“环境变量”。



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。



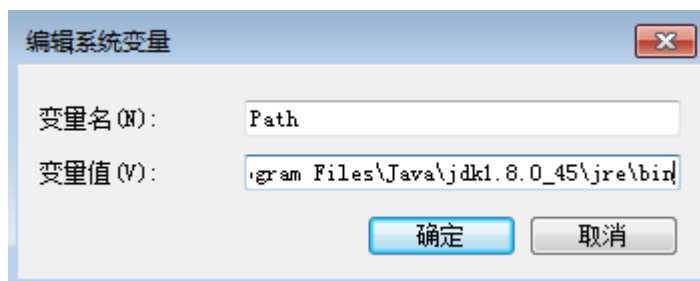


JAVA\_HOME指明JDK安装路径，配置示例：C:\ProgramFiles\Java\jdk1.8.0\_45。  
此路径下包括lib，bin等文件夹。



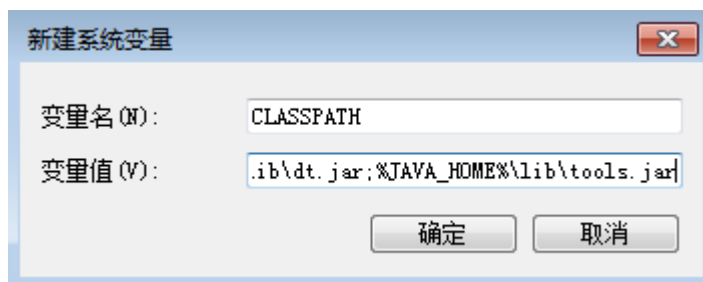
Path变量使系统可以在任何路径下识别Java命令。如果Path变量已经存在，则需在变量值最后添加路径，配置示例：;C:\Program Files\Java\jdk1.8.0\_45\bin;C:\Program Files\Java\jdk1.8.0\_45\jre\bin。

两个路径之间需要使用“;”分割，分号是英文半角。



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。

注：路径以“.”开始，表示当前路径。



5. 选择“开始 > 运行”，输入“cmd”，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。

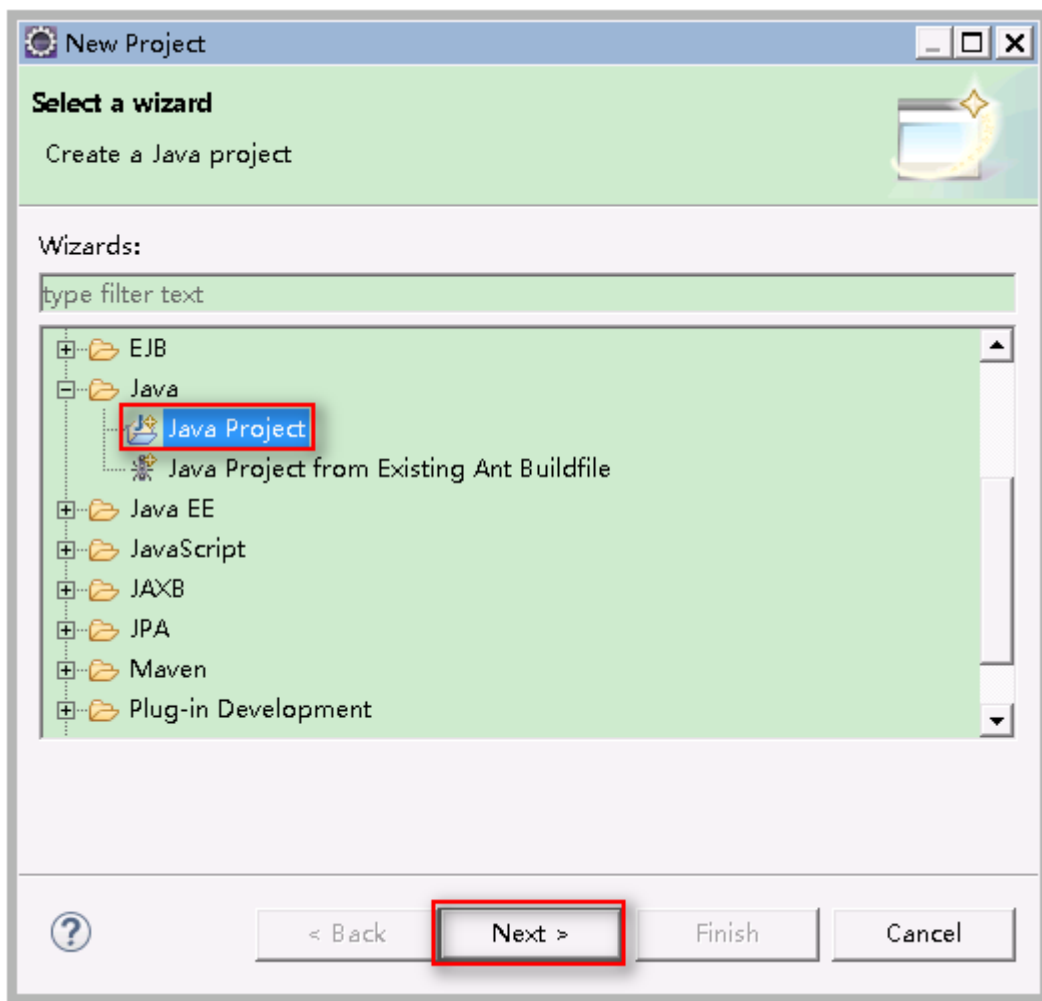
```
C:\Users\z00293999>java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b15)
Java HotSpot(TM) Client VM (build 25.45-b02, mixed mode)
```

**步骤3** 前往[Eclipse官网](#)，下载Eclipse安装包，直接解压缩到本地即可使用。

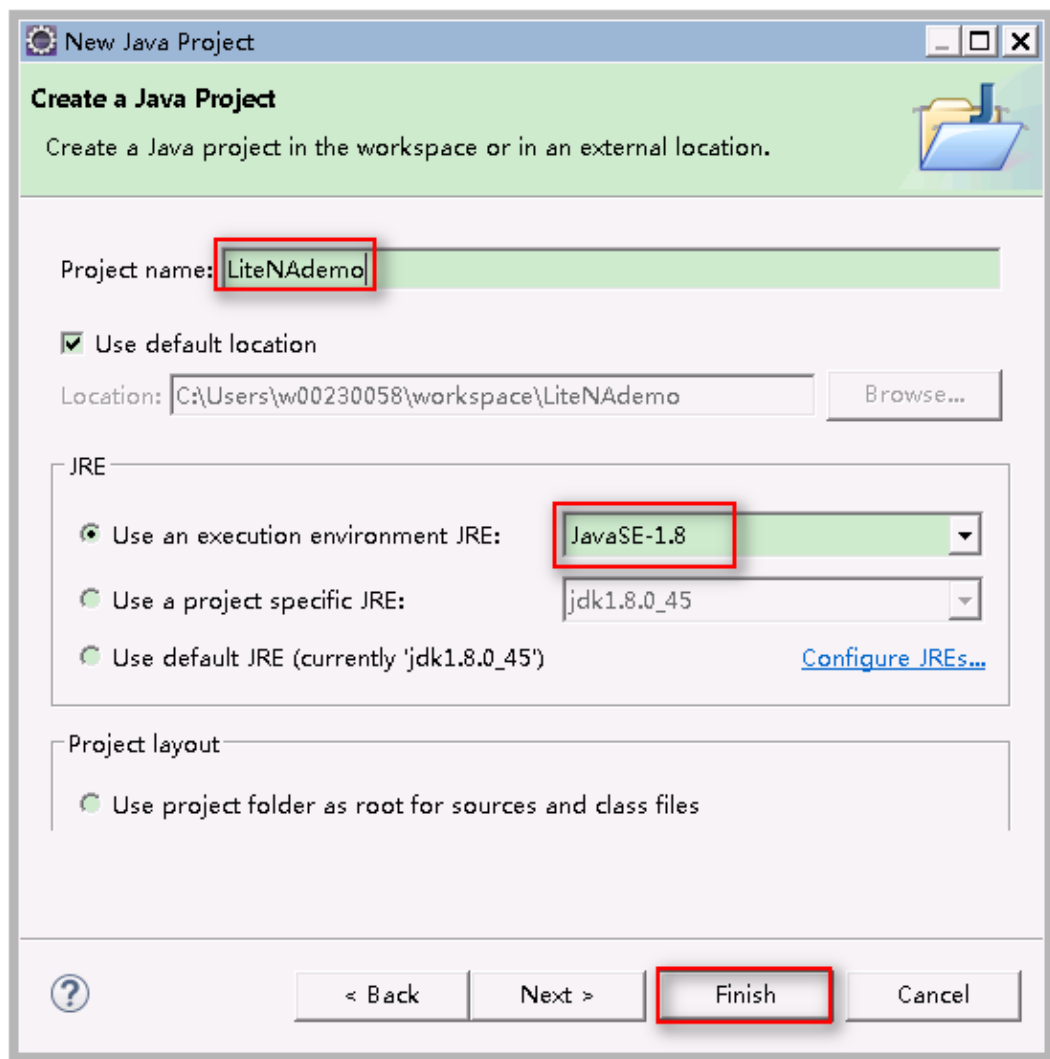
----结束

## 导入 Demo 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”。

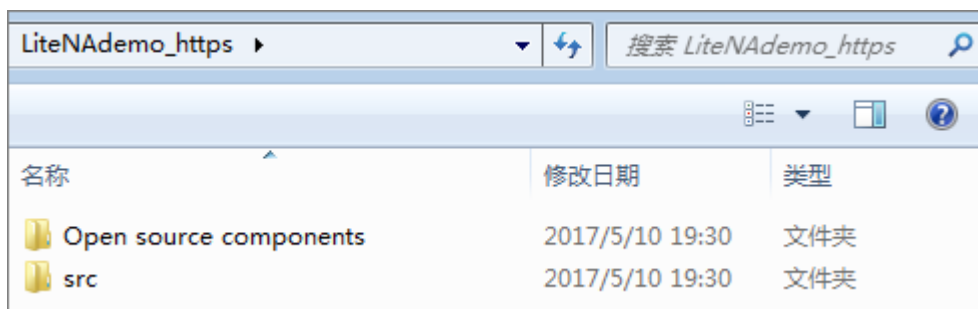


**步骤2** 填写“Project name”，JRE版本选择“JavaSE-1.8”，点击“Finish”。

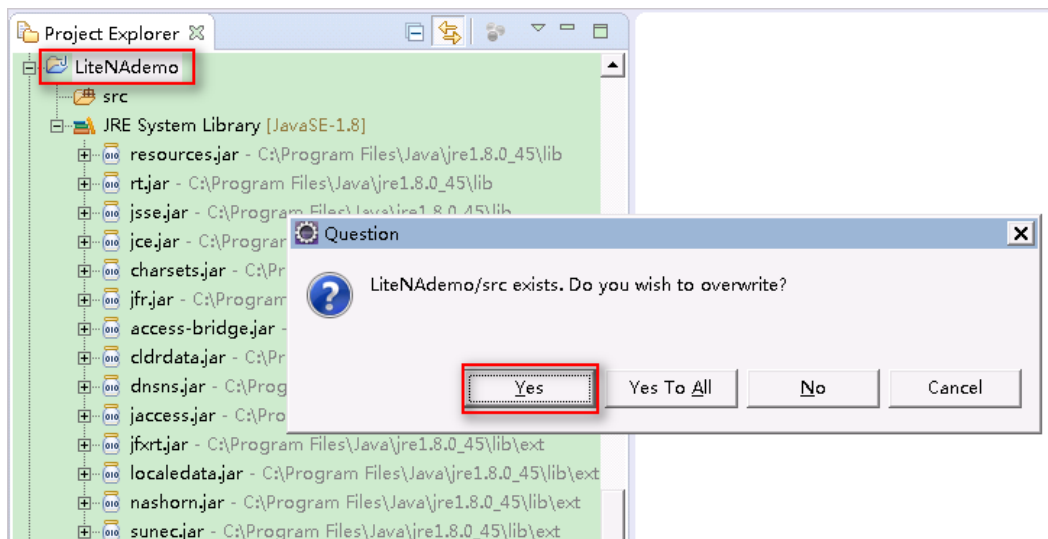


**步骤3** 下载API JAVA Demo，并解压。

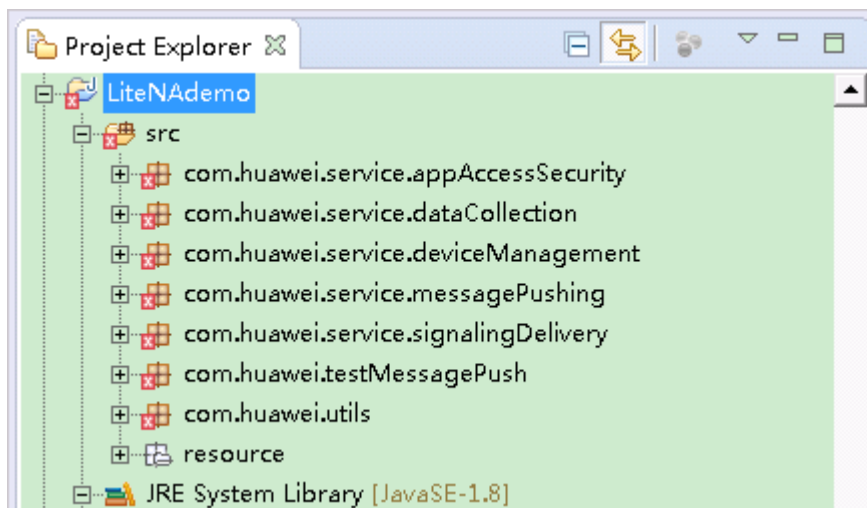
**步骤4** 完成解压后，拷贝（Ctrl+C）“Open source components”和“src”文件夹。



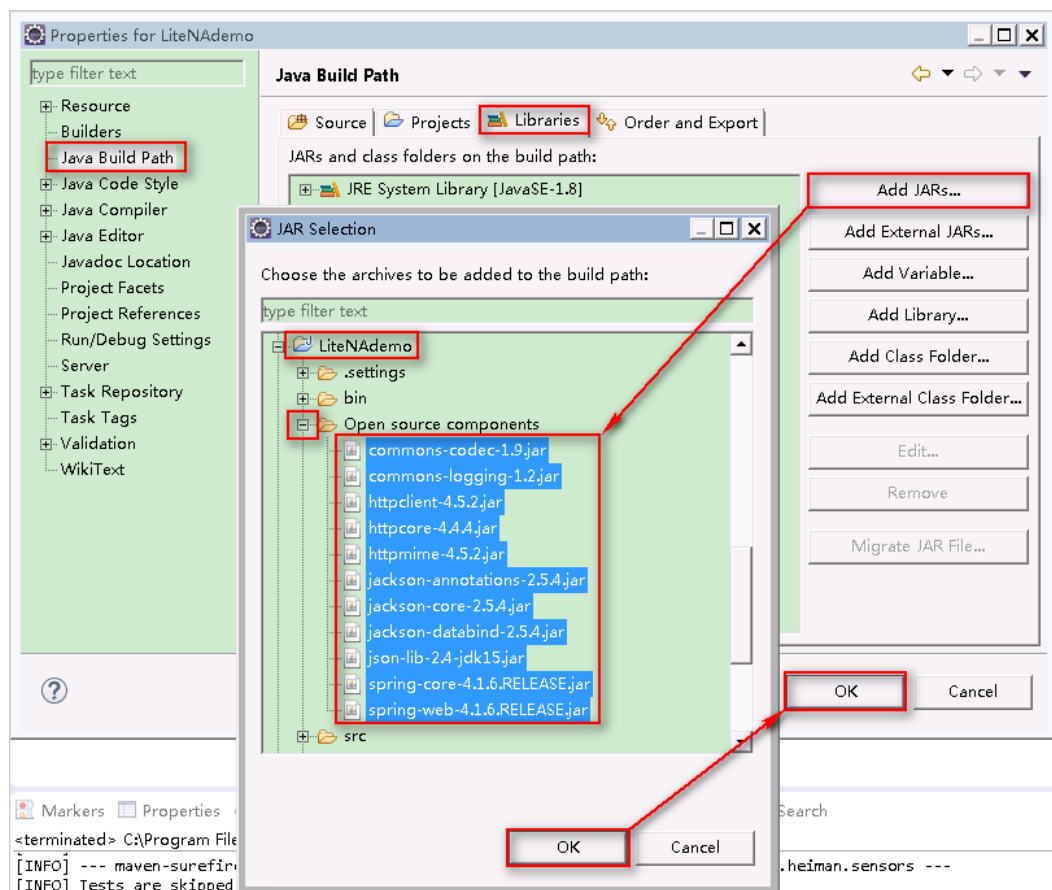
**步骤5** 打开在Eclipse创建的工程，点击选中工程名称，将拷贝的文件粘贴（Ctrl+V）到该工程目录下。



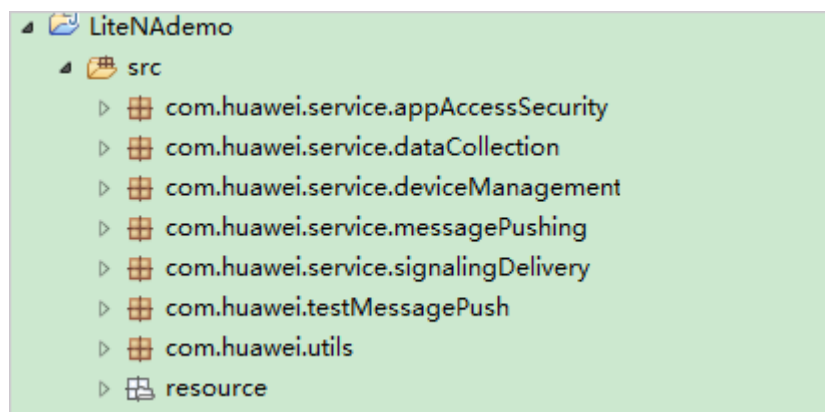
粘贴完成后，“src”目录下的文件存在错误。



**步骤6** 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”。



导入jar文件之后，“src”目录下文件的错误消失。



---结束

## 应用接入

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。

**步骤1** 在eclipse中，选择“src > com.utils > Constant.java”，修改BASE\_URL、APPID、SECRET，然后保存（Ctrl+S）。

```

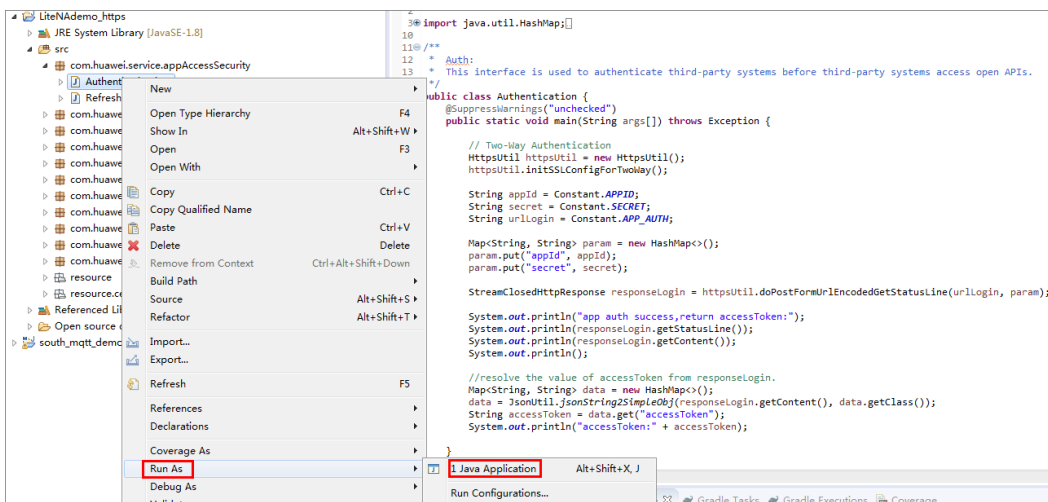
13 package com.utils;
14
15 public class Constant {
16
17     //please replace the IP and Port of the IoT platform environment address, when you use the demo.
18     public static final String BASE_URL = "https://10.10.10.99:8743";
19
20     //please replace the appId and secret, when you use the demo.
21     public static final String APPID = "HrFuCR6YS*****H0nbMdy7Waa";
22     public static final String SECRET = "QpLXCS5N*****xpDM24f5fw8a";
23
24     /*
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://10.10.10.99:9999";
29

```

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2** 在eclipse中，选择“src > com.huawei.service.appAccessSecurity”，右键单击“Authentication.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":"8d44bc6cdb0e863dfdbb8ccfed4e51","tokenType":"bearer","refreshToken":"9e11397df47877fe2afe4cd5c463621","expiresIn":3600,
accessToken:8d44bc6cdb0e863dfdbb8ccfed4e51

```

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。

----结束

## 订阅数据

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。在Java API Demo中，使用HTTP协议接收物联网平台的推送消息，不需要在物联网平台加载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。

**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://192.168.0.100:8080";
29
30 /*
31 * complete callback url.
32 * please replace url, when you use the demo.
33 */
34
35 public static final String DEVICE_ADDED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/addDevice";
36 public static final String DEVICE_INFO_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceInfo";
37 public static final String DEVICE_DATA_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceData";
38 public static final String DEVICE_DELETED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/deleteDevice";
39 public static final String MESSAGE_CONFIRM_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/commandConfirmData";
40 public static final String SERVICE_INFO_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateServiceInfo";
41 public static final String COMMAND_RSP_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/commandRspData";
42 public static final String DEVICE_EVENT_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/DeviceEvent";
43 public static final String RULE_EVENT_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/RuleEvent";
44 public static final String DEVICE_DATA_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceData";
45 public static final String DEVICE_SHADOW_MODIFIED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/modifyDeviceDesired";
46
47 public static final String SM_UPGRADE_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/upgradeSm";
48 public static final String FM_UPGRADE_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/upgradeFm";
49
50 /*
51 * Specifies the callback URL for the command execution result notification.
52 * For details about the execution result notification definition.
53 */
54 * please replace url, when you use the demo.
    
```

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

```

12 import java.util.HashMap;
13
14 /**
15  * SubscribeServiceNotification :
16  * This interface is used to subscribe service data of IoT platform.
17  */
18 public class SubscribeServiceNotification {
19
20     public static void main(String args[]) throws Exception {
21         // Two-Way Authentication
22         HttpsUtil httpsUtil = new HttpsUtil();
23         httpsUtil.setIntSSLConfigForTwoWay();
24
25         // Authentication, get token
26         String accessToken = Login(httpsUtil);
27
28         //Please make sure that the following parameter values have been modified in the Constant file.
29         String appId = Constant.APPID;
30         String urlSubscribeServiceNotification = Constant.SUBSCRIBE_SERVICE_NOTIFICATION;
31
32         /*
33          * subscribe deviceAdded notification
34          */
35         String callbackurl_deviceAdded = Constant.DEVICE_ADDED_CALLBACK_URL;
36         String notifyType_deviceAdded = Constant.DEVICE_ADDED;
37
38         Map<String, Object> paramSubscribe_deviceAdded = new HashMap<>();
39         paramSubscribe_deviceAdded.put("notifyType", notifyType_deviceAdded);
40         paramSubscribe_deviceAdded.put("callbackurl", callbackurl_deviceAdded);
41         paramSubscribe_deviceAdded.put("appId", appId);
42
43         String jsonRequest_deviceAdded = JsonUtil.toJsonObj2String(paramSubscribe_deviceAdded);
44
45         Map<String, String> header_deviceAdded = new HashMap<>();
46         header_deviceAdded.put(Constant.HEADER_APP_KEY, appId);
47         header_deviceAdded.put(Constant.HEADER_APP_AUTH, "Bearer" + " " + accessToken);
48
49         HttpResponse httpResponse_deviceAdded = httpsUtil.doPostJson(urlSubscribeServiceNotification, header_deviceAdded, jsonRequest_deviceAdded);
50
51         String bodySubscribe_deviceAdded = httpsUtil.getHttpResponseBody(httpResponse_deviceAdded);
52
53         System.out.println("SubscribeServiceNotification: " + notifyType_deviceAdded + ", response content:");
54         System.out.println(httpResponse_deviceAdded.getStatusLine());
55         System.out.println(bodySubscribe_deviceAdded);
56         System.out.println();
    
```

**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":"c5166051d466226b1fa3590d17e4a3a","tokenType":"bearer","refreshToken":"357e88ea50a45d523b7e5ff9165cf58","expiresIn":3600,"
SubscribeNotification: deviceAdded, response content:
HTTP/1.1 201 Created
    
```

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。

----结束



## 注册设备

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。

- 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。

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物联网平台删除。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":"be3ccdc5390ed14b81c85f58e2712b2","expiresIn":3600,
RegisterDirectlyConnectedDevice, response content:
HTTP/1.1 200 OK{"deviceId":"d8ac5cac-d3af-4e0c-8166-5a67e1c6f07a","verifyCode":"9999","timeout":0,"psk":"5a7f074ffdb1c46ed104a8efcaf9da0e"}
```

----结束

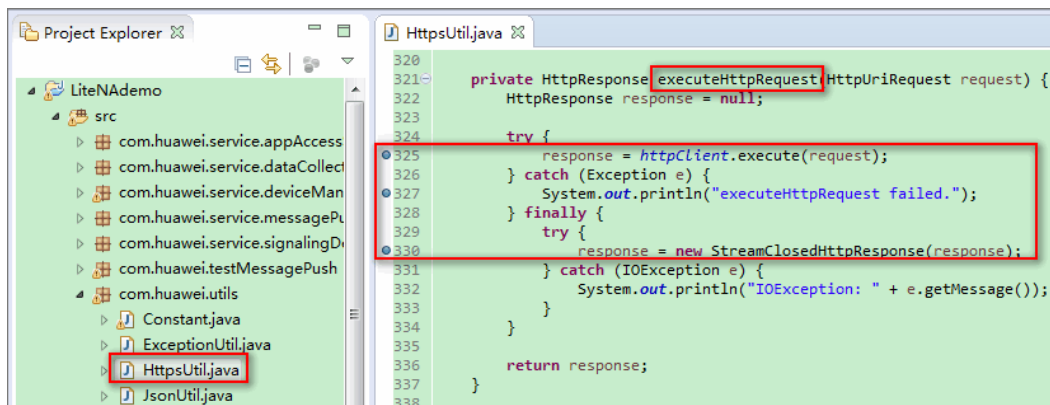
## 其他接口

请参考API参考文档完成其他接口业务的开发。

## 单步调测

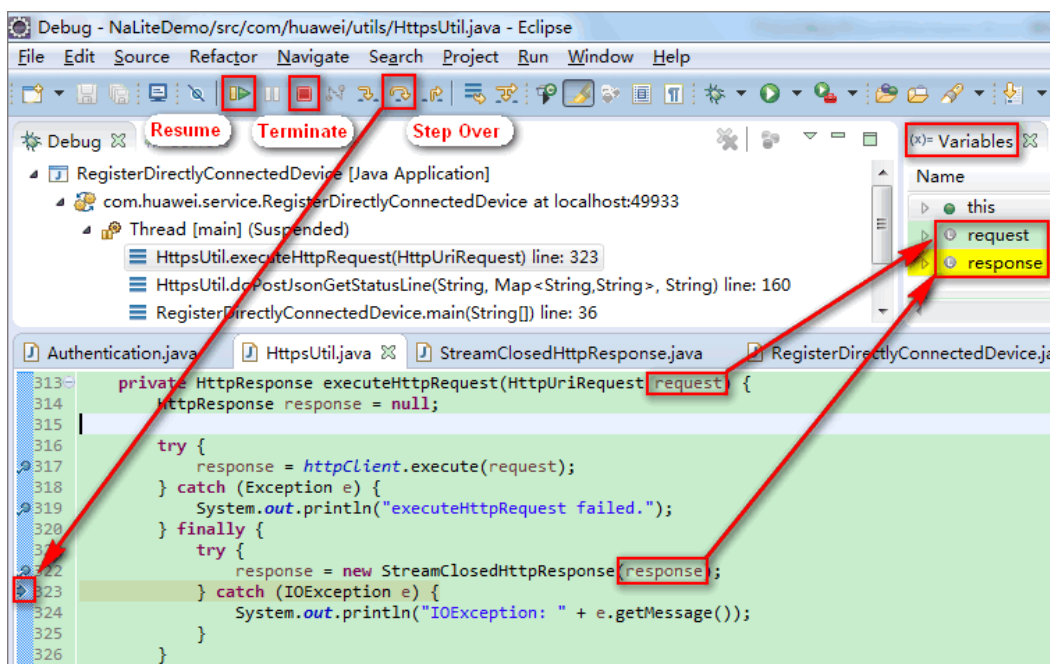
为了更直观地查看应用程序发送的消息及物联网平台的响应消息，以下方法使用了Eclipse的断点调试方法。

**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（请根据您的代码的实际情况打断点）。



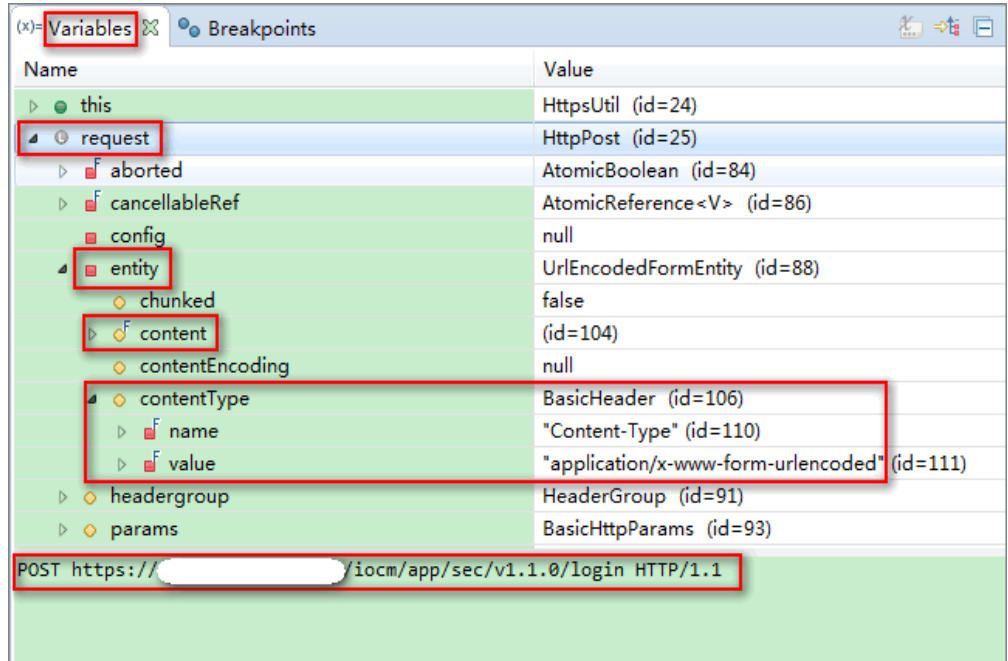
**步骤2** 右键单击需要调测的类，例如：“Authentication.java（根据您的工程类型进行选择）”，选择“Debug As > Java Application”。

**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。

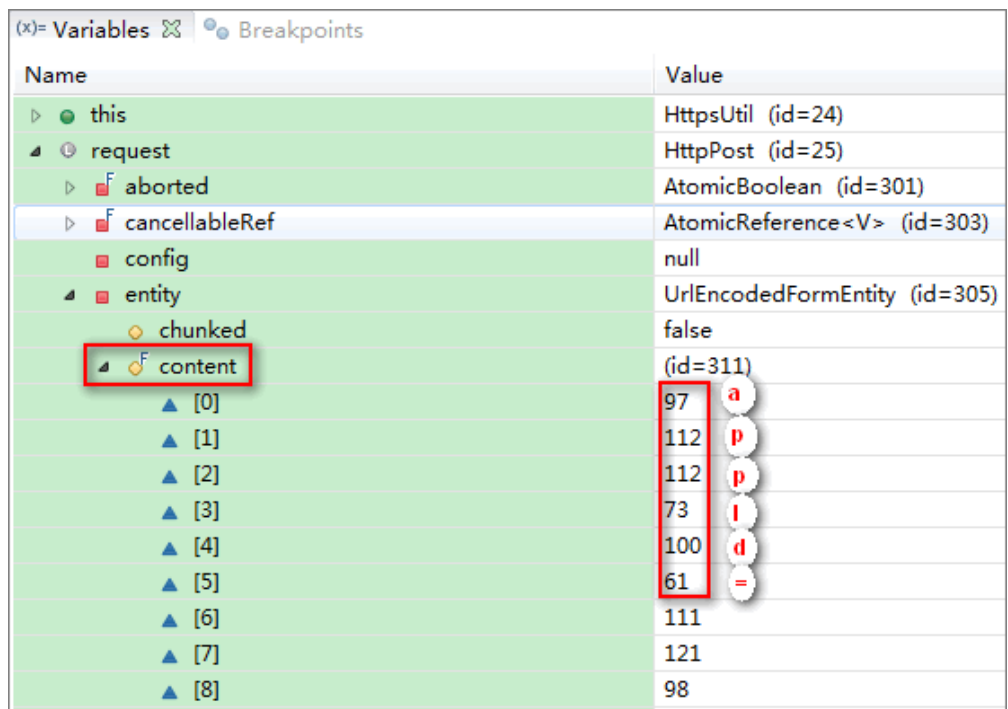


**步骤4** 在“Variables”窗口中展开“request”变量，查看请求消息的内容。

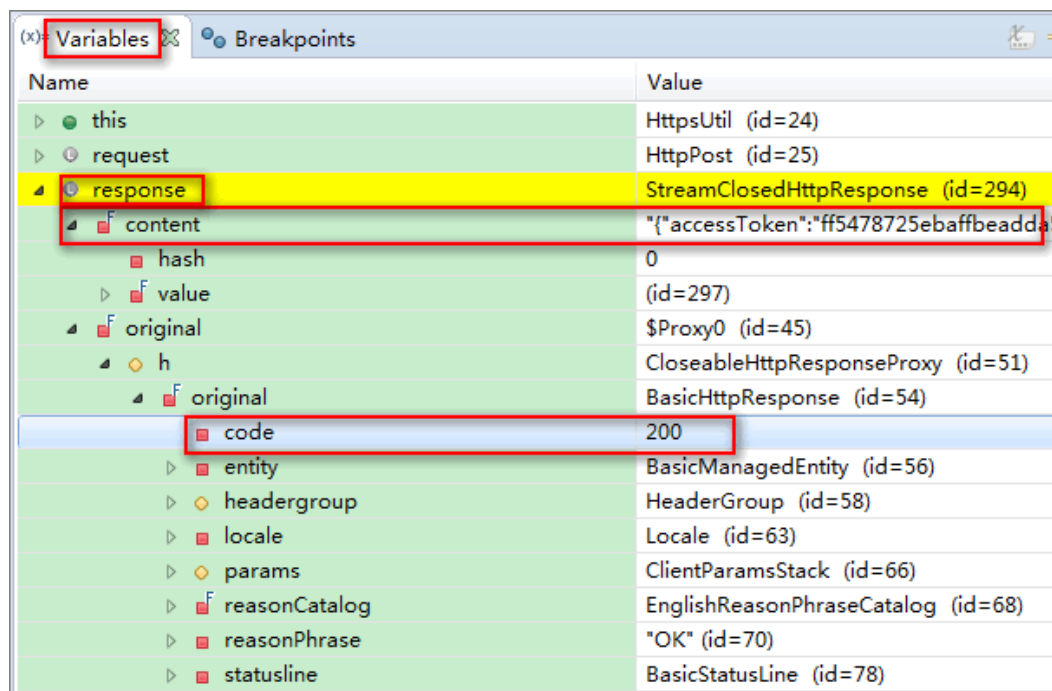
1. 选中“request”变量时，可以在下方内容展示区看到应用程序发送请求的URL；在“entity”中可以看到发送的消息内容。



- 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。



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



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测时，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

#### 4.6.1.7.2 使用 SDK 对接

##### 4.6.1.7.2.1 Java SDK 使用指南（联通用户专用）

非联通用户请查看[设备接入服务](#)。

本文以提供的应用侧Java SDK Demo为例说明如何使用JAVA SDK与物联网平台对接，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。

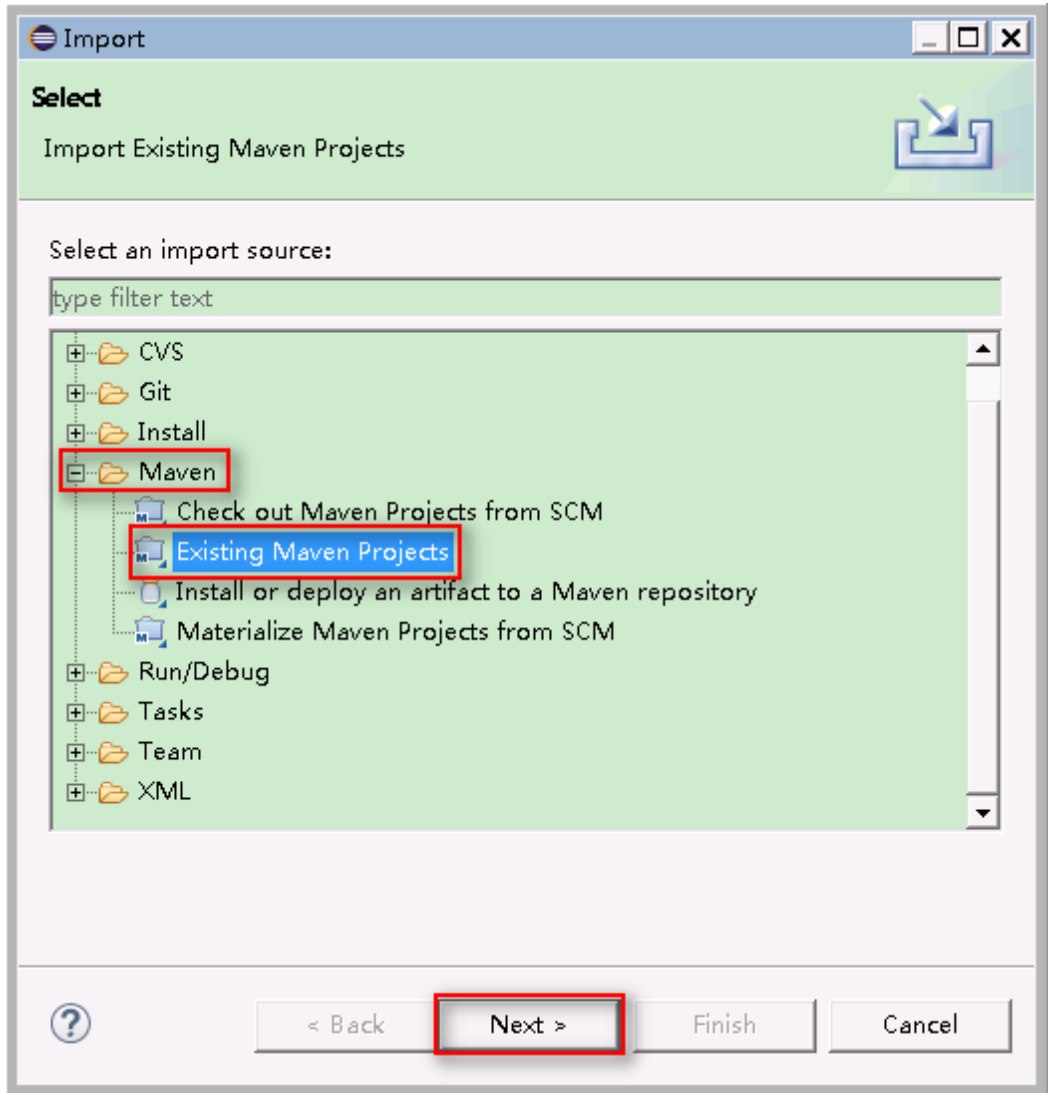
#### 开发环境要求

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)	<b>JDK 1.8</b> 及以上版本	Windows7

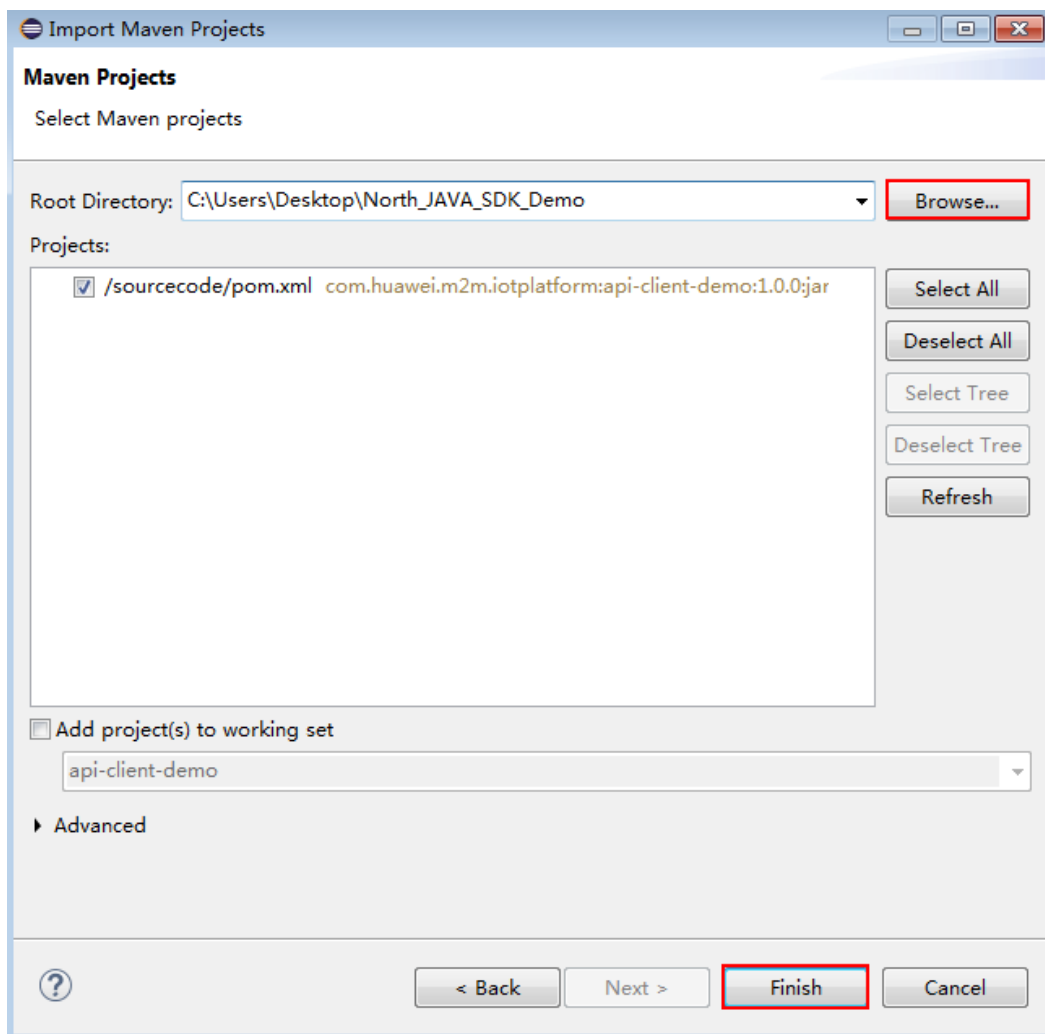
SDK包为纯JAVA的JAR包，在使用上没有特殊限制，JDK在1.8及以上版本即可。

## 导入 Demo 工程

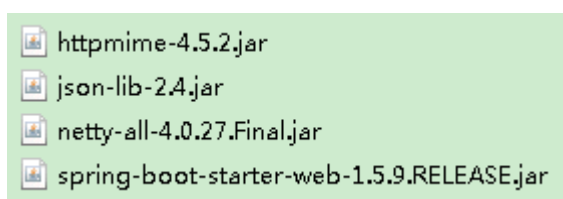
- 步骤1** 将下载的**JAVA SDK Demo**解压到本地。
- 步骤2** 打开eclipse，选择菜单“File > Import”，再选择“Maven > Existing Maven Projects”，单击“Next”。



- 步骤3** 单击“Browse”，选择demo解压后的路径，然后单击“Finish”。



**步骤4** 如果开发者无法从maven仓库下载Demo工程依赖的jar包，可以从Demo文件的components文件夹下手动导入工程依赖包。



----结束

## 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。

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。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。

- 使用测试证书:

```
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
```

- 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;
    }
});
```

### 须知

主机名校验方法应以安全为原则, 不应该直接返回true。

## 业务接口调用方法

设置好**NorthApiClient**实例后才能调用其他业务接口。以如下几个接口为例说明如何调用业务接口。

关于哪些参数需要设置, 请查看《应用侧JAVA SDK API参考》。对于可选参数, 如果业务不需要, 可以不设置或者设置为null。

### 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，不同接口的token是通用的，只用获取一次，token有效期为1小时。在即将到1小时前，可调用刷新鉴权接口重新获取token
```

```
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();  
sddInDTO.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 rddod = deviceManagement.regDirectDevice(rddInDTO, null, accessToken);  
  
//从返回的结构体rddod中获取需要的参数，如deviceId  
String deviceId = rddod.getDeviceId();
```

## 回调接口实现及证书制作

### 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  
}
```

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设置对应的回调地址。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
```

### 回调证书制作



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}
```

```
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。

- 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。

```
keytool -genkey -v -alias tomcat -keyalg RSA -keystore tomcat.keystore -validity 36500
```

```
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:
再次输入新口令:
您的名字与姓氏是什么? 输入应用服务器的IP或域名
[Unknown]: 1
您的组织单位名称是什么?
[Unknown]: 0
您的组织名称是什么?
[Unknown]: 0
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=> 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=> CN
输入 <tomcat> 的密钥口令
(如果和密钥库口令相同, 按回车):
```

**步骤3** 将物联网平台提供的根证书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。

**注：**平台的测试根证书ca.pem可以在JAVA SDK包的cert目录下找到。

```
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_ca -keystore tomcat.keystore
输入密钥库口令:
所有者: CN=> CN
发布者: CN=> 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。

```
是否信任此证书? [否]: y
证书已添加到密钥库中
[正在存储tomcat.keystore]

C:\Program Files\Java\jdk1.8.0_45\bin>
```

将物联网平台提供的根证书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。

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。



```
37 //override the callback functions if needed, otherwise, you can delete them.
38 @Override
39 public void handleDeviceAdded(NotifyDeviceAddedDTO body) {
40     System.out.println("deviceAdded ==> " + body);
41 }
42
43 @Override
44 public void handleBindDevice(NotifyBindDeviceDTO body) {
45     System.out.println("bindDevice ==> " + body);
46 }
47
48 @Override
49 public void handleDeviceInfoChanged(NotifyDeviceInfoChangedDTO body) {
```

Problems @ Javadoc Declaration Search Console

PushMessageReceiverTest [Java Application] C:\Program Files\Java\jdk1.8.0\_45\re\bin\javaw.exe (2018年7月13日 下午7:15:09)

Spring Boot (v1.5.9.RELEASE)

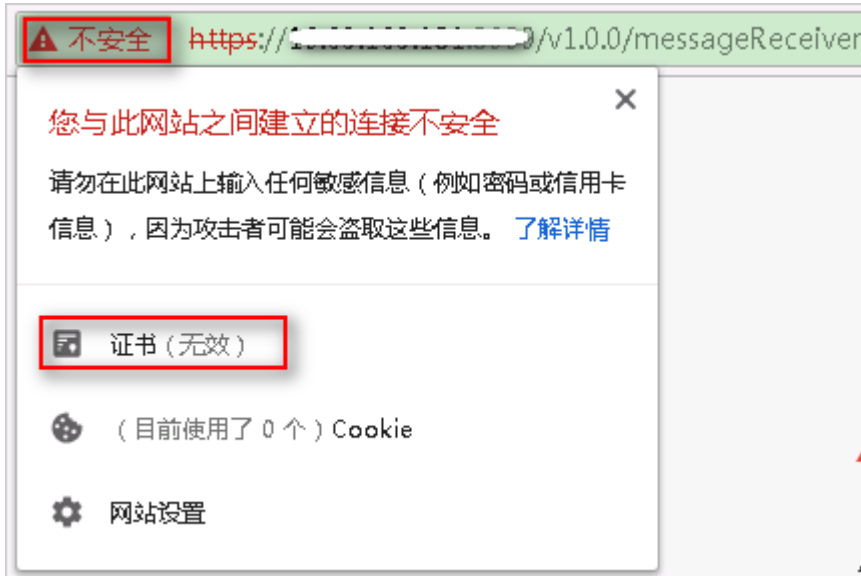
deviceAdded ==> NotifyDeviceAddedDTO [notifyType=deviceAdded, deviceId=...

----结束

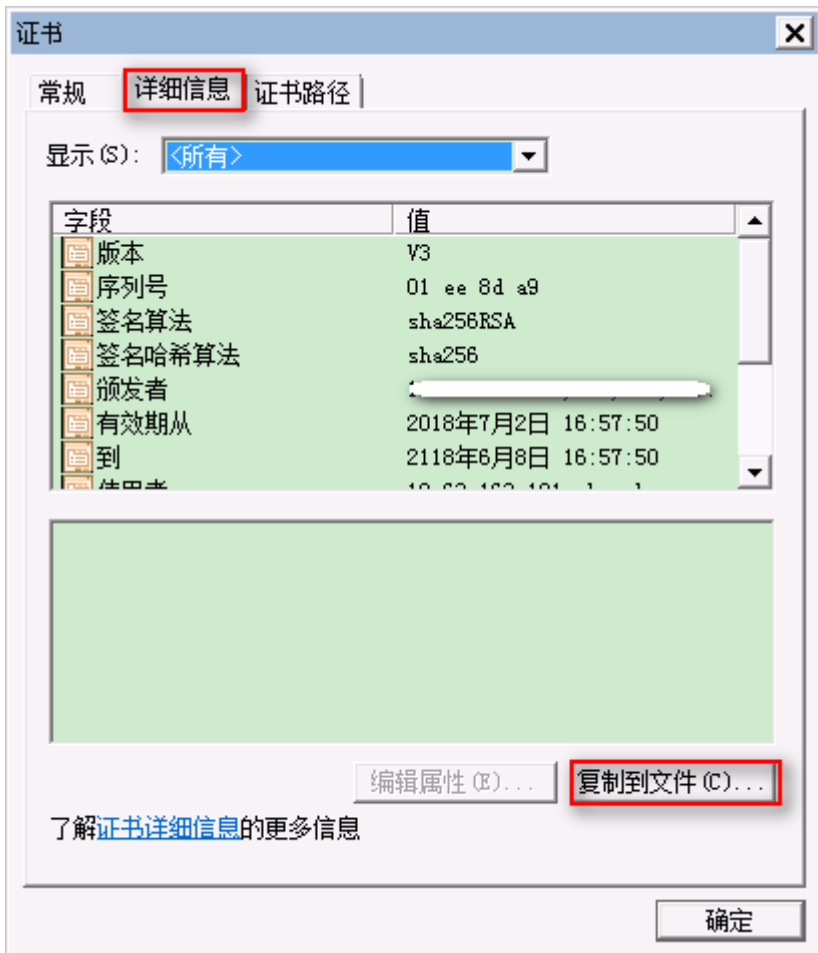
### 回调证书导出

**步骤1** 使用浏览器打开回调地址<https://server:8099/v1.0.0/messageReceiver>，以Google为例，并查看证书。

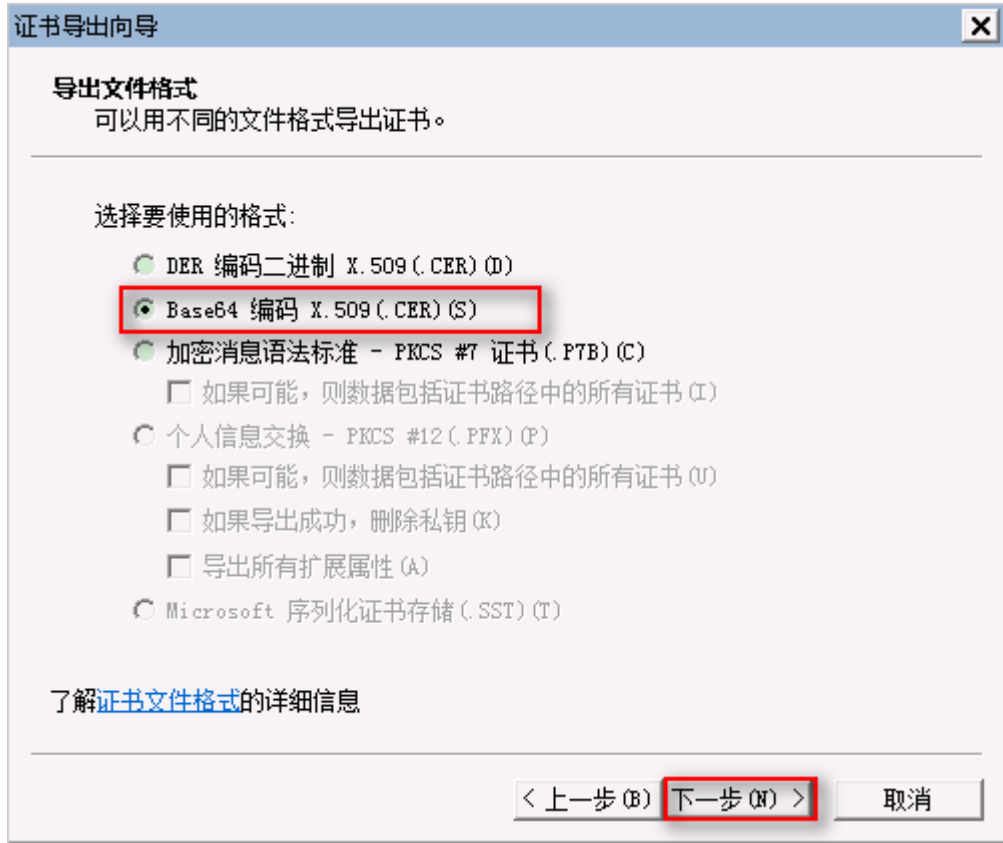
server是应用服务器的地址（即本机地址），8099是在application.properties中配置的端口。



**步骤2** 系统将弹出证书窗口，选择“详细信息”，单击“复制到文件”。



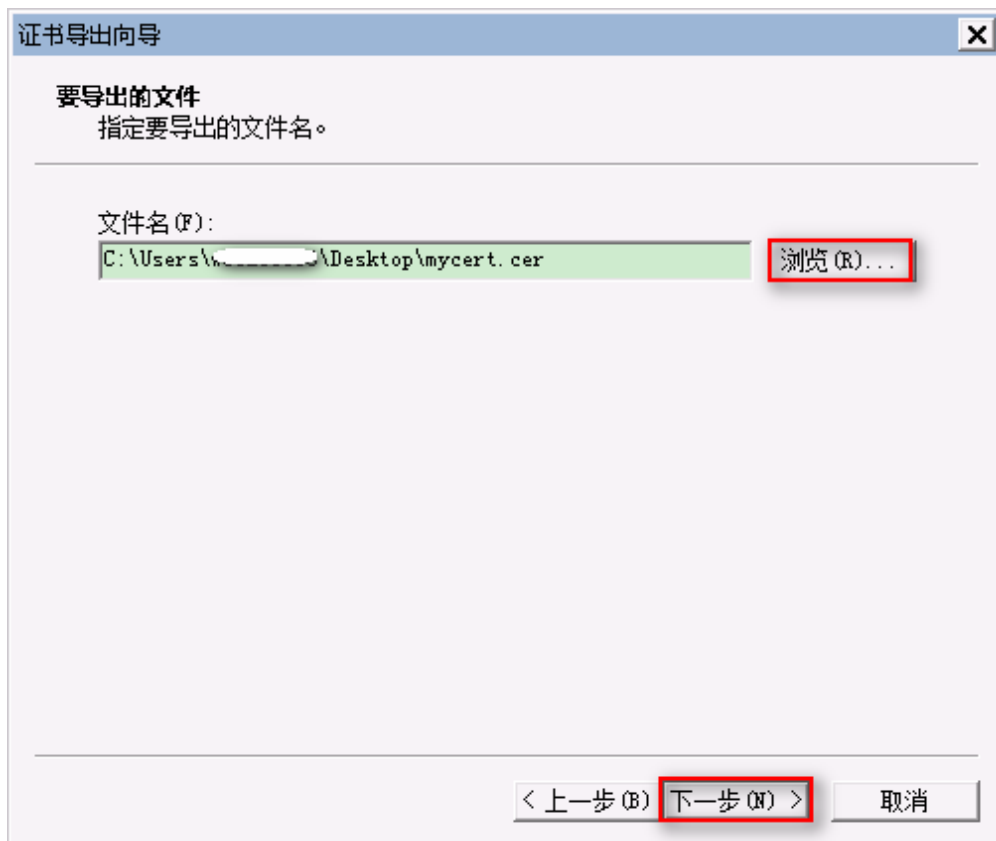
**步骤3** 单击“下一步”，进入“导出文件格式”界面，选择“Base64编码”，然后单击“下一步”。



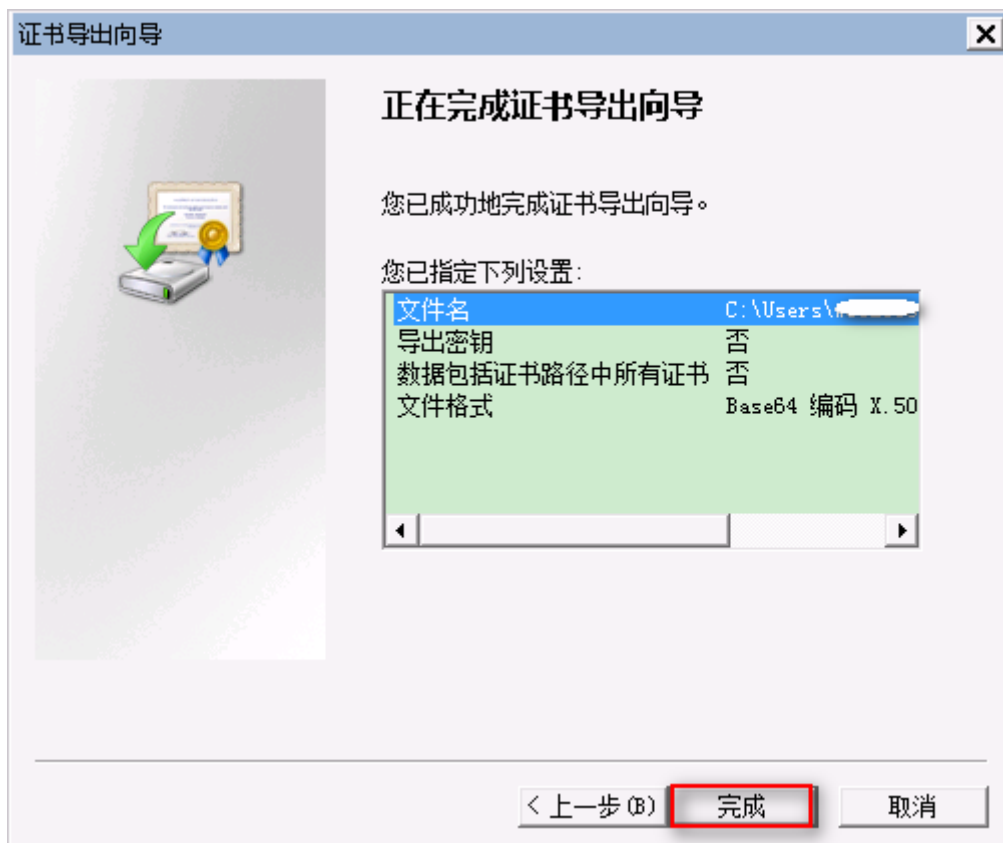
**步骤4** 指定证书的保存路径，完成证书导出。

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**。

1. 在“要导出的文件”界面，单击“浏览”，选择一个路径，输入文件名，单击“保存”，回到证书导出向导，单击“下一步”。

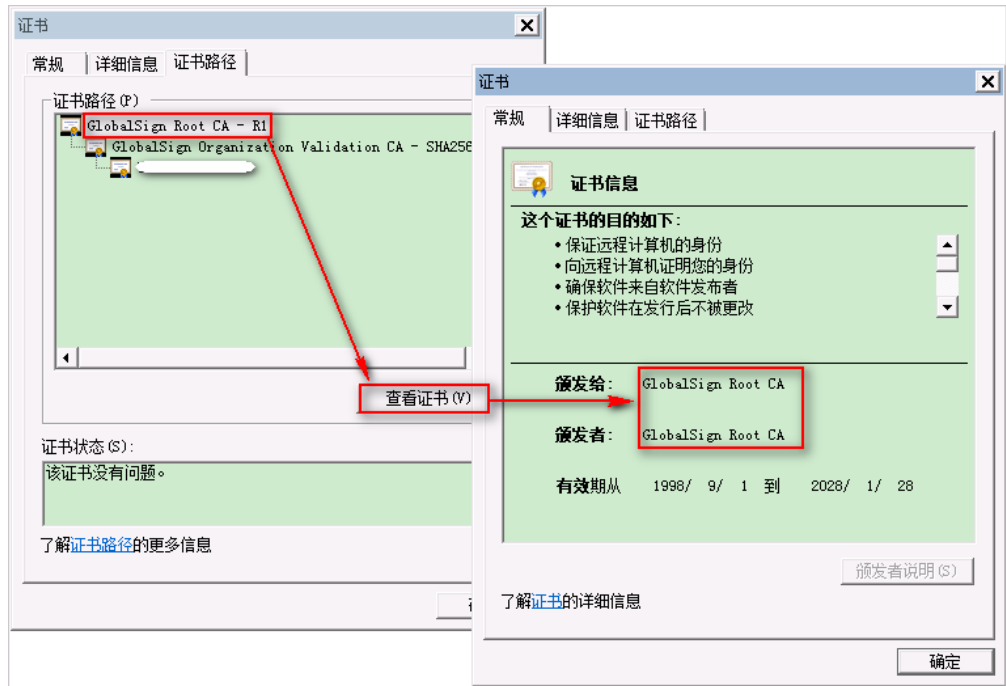


2. 单击“完成”，完成证书导出。



**步骤5** 若存在多级证书，需一一导出。

1. 在“证书”窗口，选择“证书路径”，查看多级证书，选中证书路径中的某个证书，单击“查看证书”。



2. 系统将弹出证书窗口，选择“详细信息”，然后重复上述导出证书的步骤，导出已选证书。

**步骤6** 使用文本编辑器，将所有导出的证书以首尾相连的方式，合并为一个.pem格式的文件。该文件需要上传到物联网平台相应的应用下。

```

24  5gaRQBi5+Mht39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
25  WE9gyn6CagsCqiUXObXbf+eEZSgVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXngiYZ
26  4fQRbxCl1fznQgUy286dUV4otp6F01vvpXlFQHKotw5rDgb7MzVIcbidJ4vEZV8N
27  hnacRHr2lVz2XTIIM6RUthg/aFzyQkqPOFSDX9HoLPKsEdao7WNq
28  -----END CERTIFICATE-----
29  -----BEGIN CERTIFICATE-----
30  MIIFODCCBCCgAwIBAgIQOUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
31  yjELMAkGA1UEBhMCVVMxZzAVBgNVBAoTDlZlcm1lTGFwZduLCBjbhMUMR8wHQYDVQQL
32  ExZWZlZjU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBwZlZlZjU2
33  U2lnbiwgSW5jLiAtIEZvcjBhdXR3b3JpemVkIHVzZSBvbmx5MjAwNiBwZlZlZjU2
34  ZXJpU2lnbiBDbGFzcyAzIFB1YmVzYyBQcm1tYXJ5J5IENlcnRpb2mljYXRpb24gQXV0

```

----结束

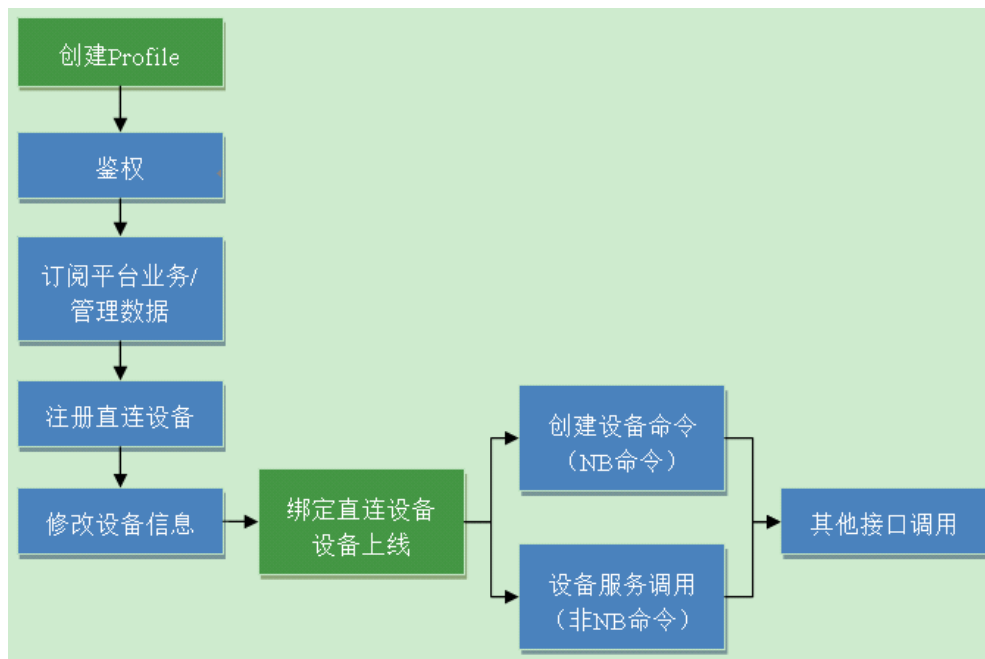
**回调证书上传**

- 步骤1 登录开发中心，进入相关项目。
- 步骤2 选择“应用 > 对接信息”，单击“证书管理”。
- 步骤3 单击“添加”，上传证书。

----结束

## 业务接口调用流程及注意事项

- 请按如下流程调用业务接口。



- 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。

服务名称	描述	最后修改时间	操作
Brightness		2018/07/16 16:28:59	

属性名称	属性类型	取值范围	步长	单位	访问模式	是否必填	操作
brightness	int	0 - 100	-	-	R	<input checked="" type="checkbox"/>	

命令名称	命令下发字段	属性类型	取值范围	步长	单位	是否必填	操作
PUT	brightness	int	0 - 100	-	-	<input checked="" type="checkbox"/>	

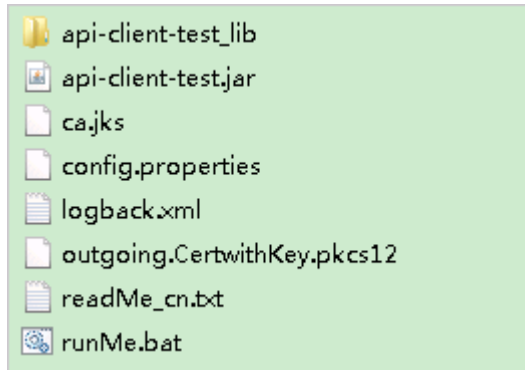
- 创建新的Profile方法：  
登录开发者中心，选择“产品 > 产品开发 > 添加 > 自定义产品”，点击“自定义产品”，进入“设置产品信息”页面。填写“产品名称”、“型号”、“厂商ID”、“所属行业”、“设备类型”、“接入应用层协议类型”等产品信息，点击“创建”。然后点击“+新建服务”（根据设备功能添加属性和命令）最后点击“保存”。
- 修改设备信息接口使用到的字段值如“设备类型”、“厂商名”、“厂商ID”、“设备型号”要与Profile定义的保持一致。

- 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。

## SDK 独立运行测试

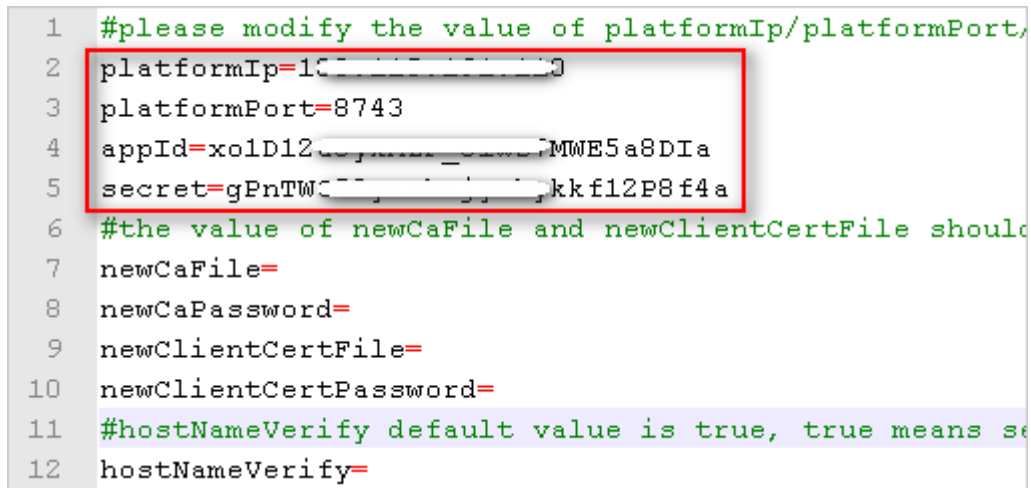
SDK包中提供了可独立运行的jar包，用于测试平台Restful接口。可独立运行测试的jar包在testSDK目录下：

图 4-1 可独立运行的 jar 包



**步骤1** 修改config.properties后再双击运行runMe.bat即可进行测试。

图 4-2 修改 config.properties



**步骤2** 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。

**步骤3** 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。

**步骤4** 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：



```

Begin test...
==> D:\O...
[!] getAuthToken() 1, get accesstoken successfully with inner certificates
AuthOutDTO {accessToken=57a37baa92cbbfca3d763ffe8e9e9d73, tokenType=bearer, refreshToken=4ba44a...
[!] refreshAuthToken() succeeded, AuthRefreshOutDTO {accessToken=309c198bc768ca31e83996936fba2...
3762bcf2}
[!] regDirectDevice() succeeded, DirectDeviceRegOutDTO {deviceId=ef72daa2-772d-4a9b-a2bc-71a48a...
[!] modifyDeviceInfo() succeeded
[!] deleteDevice() succeeded
    
```

----结束

#### 4.6.1.7.2.2 Python SDK 使用指南（联通用户专用）

非联通用户请查看[设备接入服务](#)。

本文以提供的应用侧Python SDK Demo为例说明如何使用Python SDK与物联网平台对接。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。

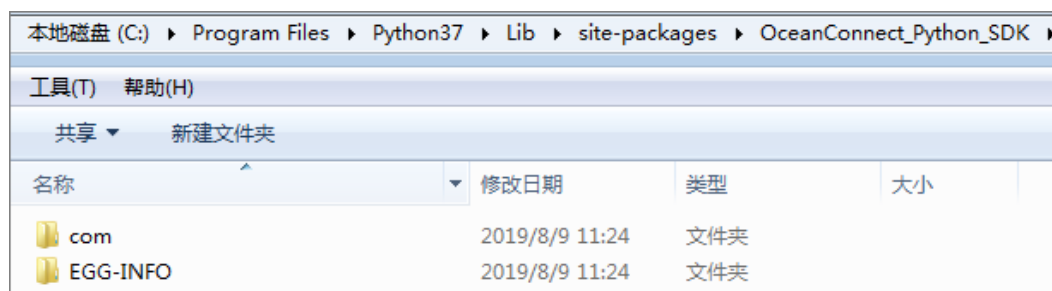
#### 开发环境要求

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。

#### 导入 Demo 工程

**步骤1** 将Python SDK解压至本地Python安装路径的“Lib\site-packages”目录下，如，C:\Program Files\Python37\Lib\site-packages。

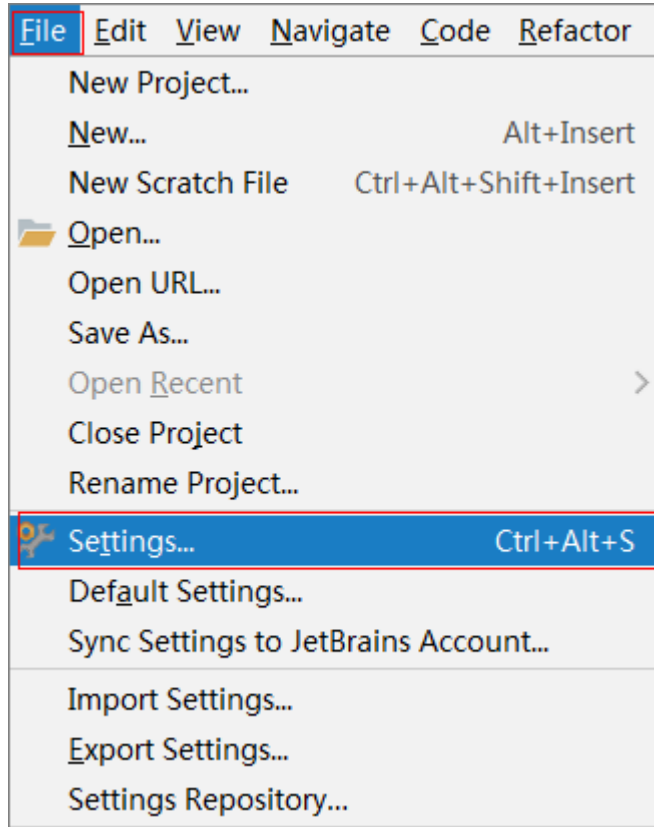


**步骤2** 将Python SDK Demo解压到本地。

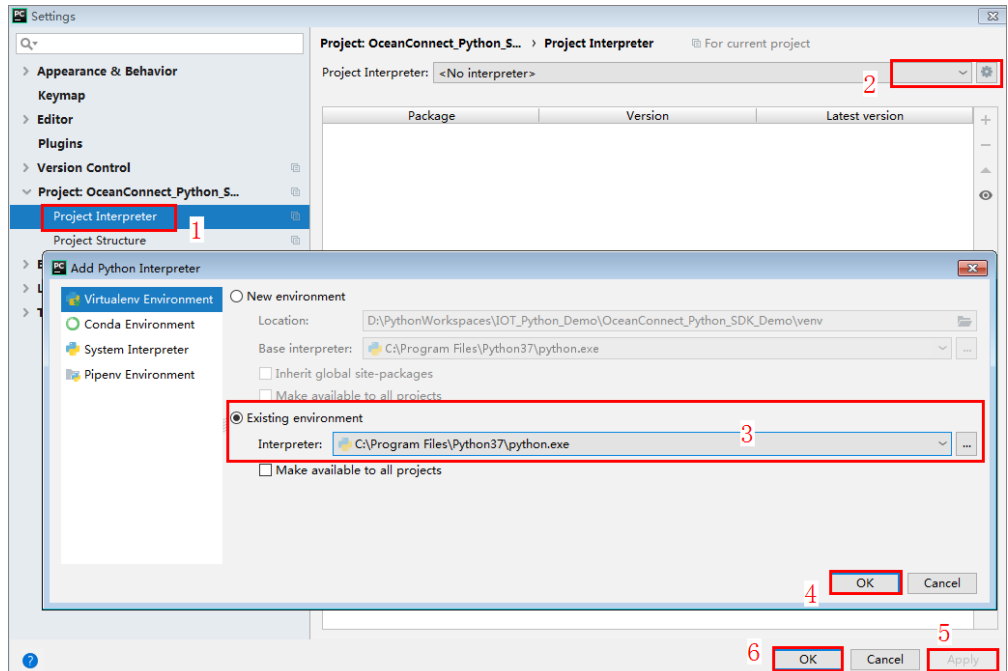
**步骤3** 打开PyCharm，选择“Open”，再选择Demo解压后的路径，单击“OK”。

**步骤4** 配置Project Interpreter。

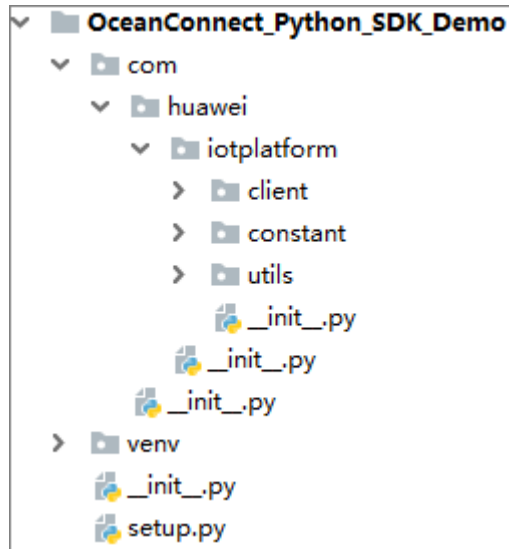
1. 选择“File > Settings > Project: OceanConnect \_ Python \_ SDK \_ Demo > Project Interpreter”。



2. 选择SDK所在的解释器，如，C:\Program Files\Python37\python.exe，单击“Apply”，最后单击“OK”即可正确配置工程所需要的SDK。



**步骤5** Python SDK Demo目录结构如下，Demo工程中已包含SDK库。



----结束

## 配置平台信息及证书

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、appld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。

```
[CLIENT_INFO]
platformIp = 100.100.100.100
platformPort = 8743
appld = FT8EjQ8O****VbW60Qb8xvga
secret = TtuK4Paf****yAv66himUk8a
```

在“Constant.py”中，读取证书和CLIENT\_INFO。

```
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")
    appld = cf.get("CLIENT_INFO", "appld")
    secret = cf.get("CLIENT_INFO", "secret")
    return platformIp, platformPort, appld, secret

# 设置平台对接信息
def clientInfo(self):
    clientInfo = ClientInfo()
    clientInfo.setPlatformIp((Constant().readConfFile())[0])
    clientInfo.setPlatformPort((Constant().readConfFile())[1])
    clientInfo.setAppld((Constant().readConfFile())[2])
```

```
clientInfo.setSecret((Constant().readConfFile())[3])
clientInfo = DictUtil.dto2dict(clientInfo)
return clientInfo
```

## 业务接口调用方法

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**”中。以如下几个接口为例说明如何调用业务接口：

关于哪些参数需要设置，请查看《应用侧Python SDK API参考》对于可选参数，如果业务不需要，可以不设置。

### 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=os.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")
```

## 回调接口实现及证书制作

### 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
```

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设置对应的回调地址。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。

## 回调证书制作

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。

**注：**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1** 打开windows命令行窗口，输入where openssl，查询是否安装openssl。

```
where openssl
```

```
C:\Users\>where openssl
D:\>openssl\GnuWin32\bin\openssl.exe
C:\Users\>
```

**步骤2** 创建测试CA证书。

1. 创建私钥。  
openssl genrsa -out ca-key.pem 1024
2. 创建证书请求。  
openssl req -new -out ca-req.csr -key ca-key.pem
3. 自签署证书。  
openssl x509 -req -in ca-req.csr -out ca-cert.pem -signkey ca-key.pem -days 3650

```
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 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 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。

1. 创建私钥。  
openssl genrsa -out server.key 1024
2. 创建证书请求。  
openssl req -new -out server-req.csr -key server.key
3. 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

```

root@SZ100480858 /usr/sakeCert# openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
root@SZ100480858 /usr/sakeCert# 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@SZ100480858 /usr/sakeCert# 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=C=cn,ST=gd,L=sz,O=test,OU=test,CN=test,emailAddress=test
Getting Private key
Getting CA Private Key
root@SZ100480858 /usr/sakeCert#
    
```

**步骤4 生成client证书。**

1. 创建私钥。  
openssl genrsa -out client.key 1024
2. 创建证书请求。  
openssl req -new -out client-req.csr -key client.key
3. 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

```

root@SZ100480858 /usr/sakeCert# openssl genrsa -out client.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
root@SZ100480858 /usr/sakeCert# 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@SZ100480858 /usr/sakeCert# 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=C=cn,ST=gd,L=sz,O=test,OU=test,CN=test,emailAddress=test
Getting Private key
Getting CA Private Key
root@SZ100480858 /usr/sakeCert#
    
```

**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”。

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。

```

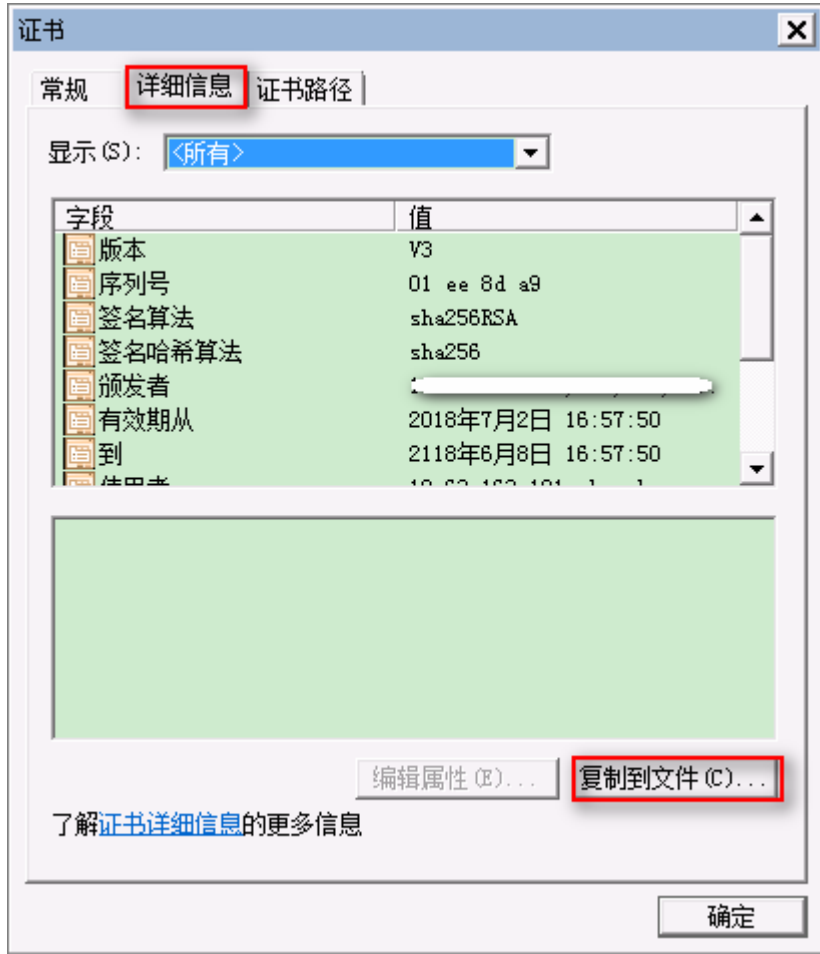
class PushMessageReceiver(object):

    def handleDeviceAdded(self):
        print("deviceAdded ==> ", request.json)
        pass

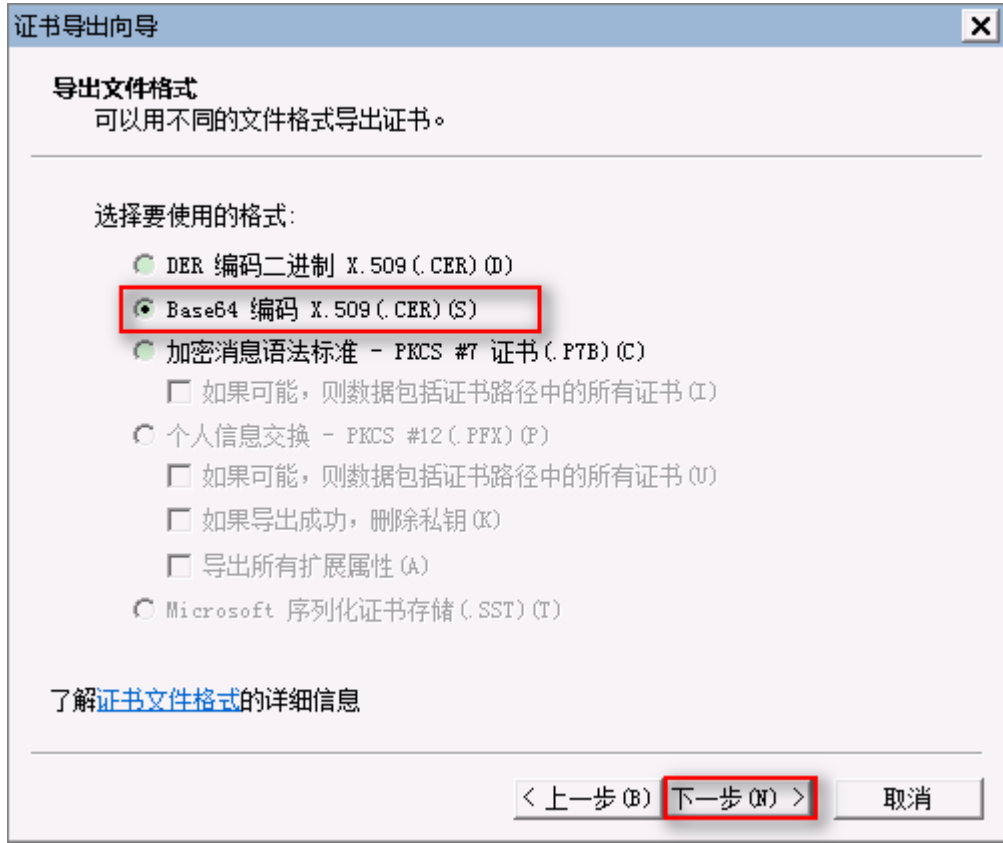
    def handleBindDevice(self):
        print("bindDevice ==> ", request.json)
        pass
    
```







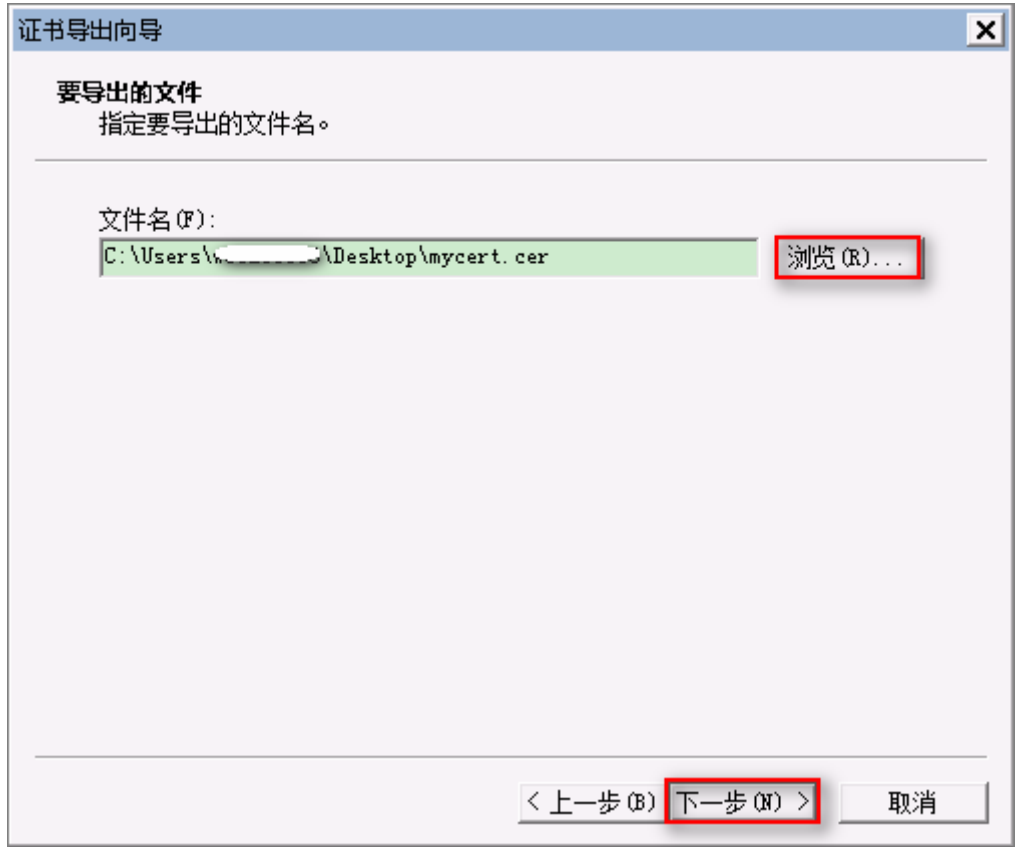
**步骤3** 单击“下一步”，进入“导出文件格式”界面，选择“Base64编码”，然后单击“下一步”。



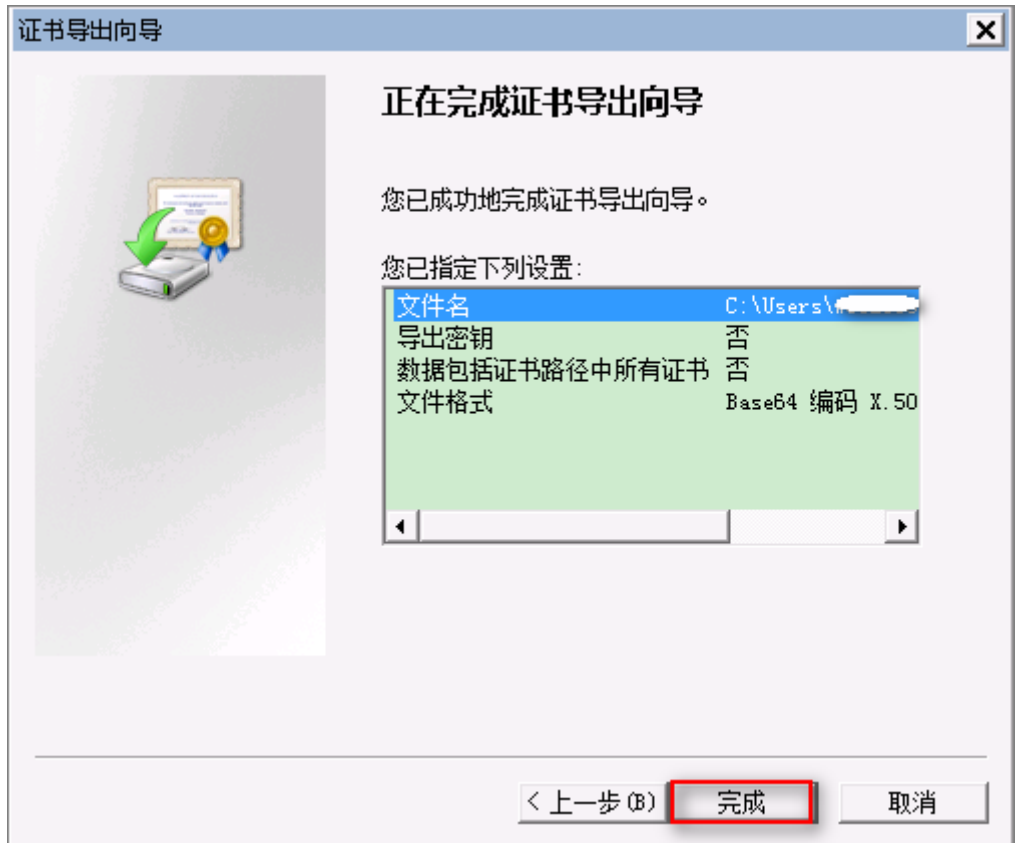
**步骤4** 指定证书的保存路径，完成证书导出。

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**。

1. 在“要导出的文件”界面，单击“浏览”，选择一个路径，输入文件名，单击“保存”，回到证书导出向导，单击“下一步”。

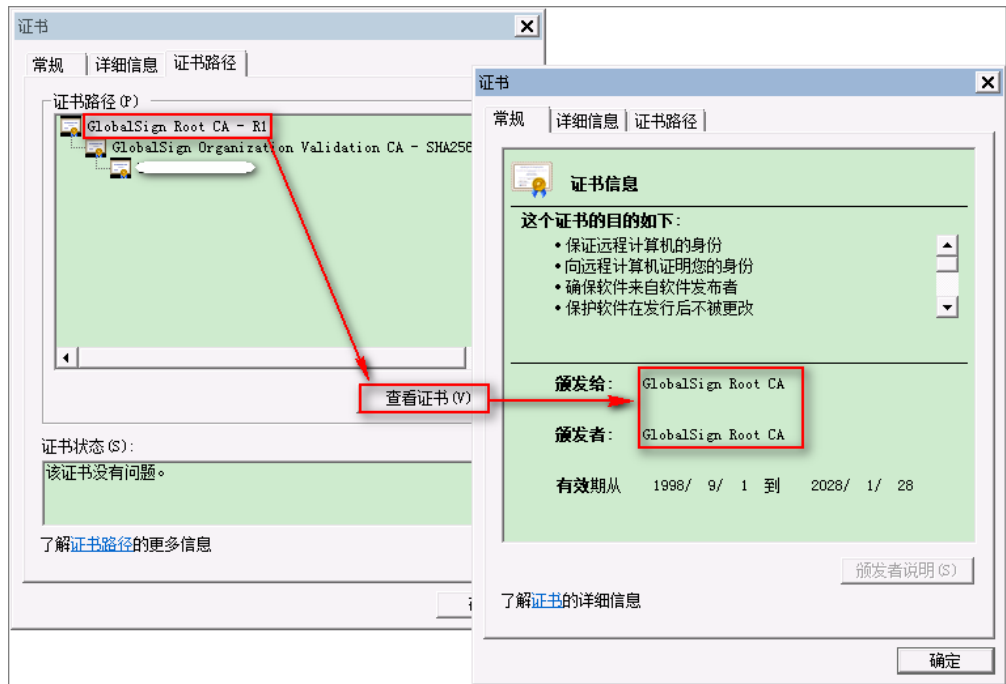


2. 单击“完成”，完成证书导出。



**步骤5** 若存在多级证书，需一一导出。

1. 在“证书”窗口，选择“证书路径”，查看多级证书，选中证书路径中的某个证书，单击“查看证书”。



2. 系统将弹出证书窗口，选择“详细信息”，然后重复上述导出证书的步骤，导出已选证书。

**步骤6** 使用文本编辑器，将所有导出的证书以首尾相连的方式，合并为一个.pem格式的文件。该文件需要上传到物联网平台相应的应用下。

```

24 5gaRQBi5+Mht39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6Bw1K
25 WE9gyn6CagsCqiUXObXbf+eEZSgVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
26 4fQRbxCl1fznQgUy286dUV4otp6F01vvpX1FQHK0tw5rDgb7MzVIcbidJ4vEZV8N
27 hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
28 -----END CERTIFICATE-----
29 -----BEGIN CERTIFICATE-----
30 MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
31 yjELMAkGA1UEBhMCVVMxPzAVBgNVBAoTDlZlcm1TaWduLCBjbmMuMR8wHQYDVQQL
32 ExZWZlZjU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBwZWZl
33 U2lnbiwgSW5jLiAtIEZvcjBhdXR0b3JpemVkIHVzZS8vbm90aW4uYykgMjAwNiBw
34 ZXJpU2lnbiBDbGFzcyAzIFB1Ym90aW4uYykgMjAwNiBwZWZlZjU2lnbiwgSW5jLiAt
  
```

----结束

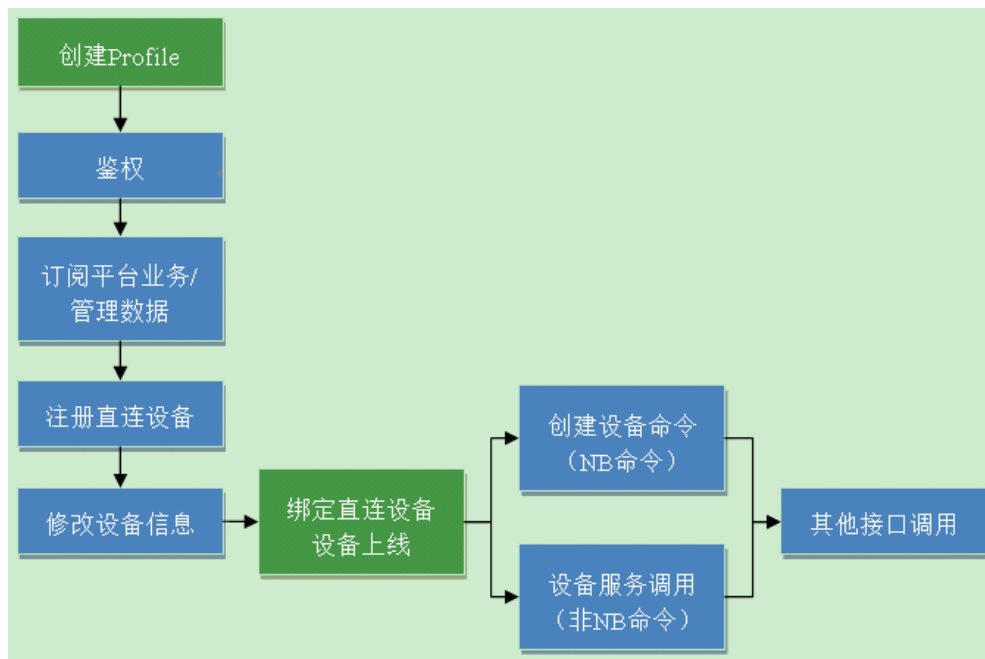
### 回调证书上传

- 步骤1 登录开发中心，进入相关项目。
- 步骤2 选择“应用 > 对接信息”，单击“证书管理”。
- 步骤3 单击“添加”，上传证书。

----结束

## 业务接口调用流程及注意事项

- 请按如下流程调用业务接口。



- 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。



- 创建新的Profile方法：  
登录开发者中心，选择“产品 > 产品开发 > 添加 > 自定义产品”，点击“自定义产品”，进入“设置产品信息”页面。填写“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“创建”。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“保存”。
- 修改设备信息接口使用到的字段值如“设备类型”、“厂商名”、“厂商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，具体信息可参考《Python SDK API参考文档》中“应用安全接入 > 定时刷新token”章节的说明。

# 5 调测证书制作（联通用户专用）

非联通用户请查看[设备接入服务](#)。

调测证书，又叫做自签名证书，用于客户端通过HTTPS访问服务端时进行安全认证。在物联网平台的使用中，可用于物联网平台向应用服务器采用HTTPS协议推送数据时，物联网平台认证应用服务器的合法性。本文以Windows环境为例，介绍通过Openssl工具制作调测证书的方法，生成的证书为PEM编码格式的证书，后缀为.cer。

常见的证书存储格式如下表所示。

存储格式	说明
DER	二进制编码，后缀名 <code>.der/.cer/.crt</code>
PEM	BASE 64编码，后缀名 <code>.pem/.cer/.crt</code>
JKS	Java的证书存储格式，后缀名 <code>.jks</code>

## 📖 说明

自签名证书仅用于调测阶段，在商用时，您需要向知名CA机构申请证书，否则可能会带来安全风险。

**步骤1** 在浏览器中访问<https://slproweb.com/products/Win32OpenSSL.html>，下载并进行安装OpenSSL工具。

**步骤2** 以管理员身份运行cmd命令行窗口。

**步骤3** 执行`cd c:\openssl\bin`（请替换为openssl实际安装路径），进入openssl命令视图。

**步骤4** 执行如下命令生成CA根证书私钥文件`ca_private.key`。

```
openssl genrsa -passout pass:123456 -aes256 -out ca_private.key 2048
```

- aes256：代表加密算法。
- passout pass：代表私钥密码。
- 2048：代表密钥长度。

**步骤5** 执行如下命令使用CA根证书私钥文件生成csr文件`ca.csr`，用于**步骤6**生成CA根证书。

```
openssl req -passin pass:123456 -new -key ca_private.key -out ca.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=IoT/CN=CA"
```

如下信息您可以根据实际情况进行修改。

- C: 代表国家, 填写CN。
- ST: 地区, 如GD。
- L: 城市, 如SZ。
- O: 组织, 如Huawei。
- OU: 组织单位, 如IoT。
- CN: Common Name, 填写为CA的组织名, 如CA。

**步骤6** 执行如下命令生成CA根证书ca.cer。

```
openssl x509 -req -passin pass:123456 -in ca.csr -out ca.cer -signkey ca_private.key -CAcreateserial -days 3650
```

如下信息您可以根据实际情况进行修改。

- passin pass: 必须与**步骤4**中设置的私钥密码保持一致。
- days: 代表证书有效期。

**步骤7** 执行如下命令生成应用服务器端私钥文件。

```
openssl genrsa -passout pass:123456 -aes256 -out server_private.key 2048
```

**步骤8** 执行如下命令生成应用服务器端csr文件, 用于生成服务端证书。

```
openssl req -passin pass:123456 -new -key server_private.key -out server.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=IoT/CN=appserver.iot.com"
```

如下信息您可以根据实际情况进行修改。

- C: 代表国家, 填写CN。
- ST: 地区, 如GD。
- L: 城市, 如SZ。
- O: 组织, 如Huawei。
- OU: 组织单位, 如IoT。
- CN: Common Name, 一般填写为应用服务器的域名或IP。

**步骤9** 通过CA私钥文件ca\_private.key对服务端csr文件server.csr进行签名, 生成服务端证书文件server.cer。

```
openssl x509 -req -passin pass:123456 -in server.csr -out server.cer -sha256 -CA ca.cer -CAkey ca_private.key -CAserial ca.srl -CAcreateserial -days 3650
```

**步骤10** (可选) 如果您需要.crt/.pem后缀的证书, 可以根据如下命令进行转换。下面将以server.cer转为为server.crt为例进行说明, 需要转换ca.cer证书时, 请将命令中的server替换为ca。

```
openssl x509 -inform PEM -in server.cer -out server.crt
```

**步骤11** 在openssl安装目录的bin文件夹下, 获取生成的CA证书 ( ca.cer/ca.crt/ca.pem )、应用服务器证书 ( server.cer/server.crt/server.pem ) 和私钥文件 ( server\_private.key )。其中CA证书用于加载到物联网平台, 应用服务器证书和私钥文件用于加载到应用服务器。

----结束



## 加载证书

物联网平台采用HTTPS协议向应用服务器推送消息时，需要在物联网平台上加载CA证书，下面将详细介绍在开发中心上传证书的方法。在设备管理服务控制台上加载证书请参考[加载推送证书](#)。

**步骤1** 登录开发中心控制台，进入到开发中心。

**步骤2** 选择“应用 > 对接信息”，在“推送证书”区域，点击“证书管理”。



**步骤3** 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4** 系统弹出“上传证书”窗口，选择证书文件，并填写“域名/IP与端口”，其它信息不用修改。点击“上传”。

### 上传CA证书 ✕

**\*上传证书文件**

文件大小不超过1M，且必须为pem文件 点击选择文件

**\*域名/IP与端口**

请输入域名/IP与端口

例如：api.ct10649.com:9001或127.0.1.1:8080  
请输入合法的域名/IP及端口

**\*lb昵称**

default

是否检查CNAME ?

上传 取消

----结束

# 6 自助测试（联通用户专用）

非联通用户请查看[设备接入服务](#)。

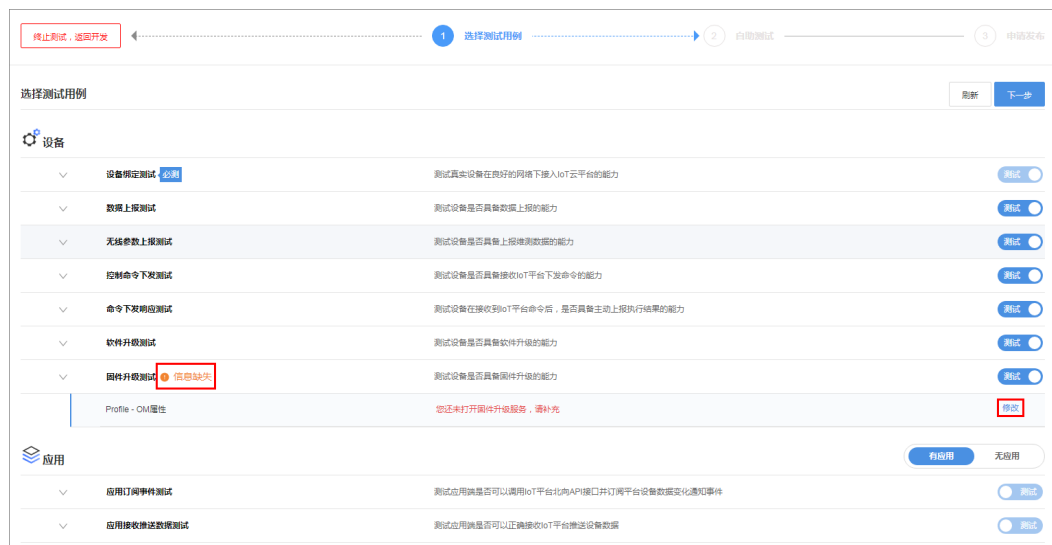
自助测试提供了端到端的测试用例，帮助开发者自助完成产品的基础能力测试，如数据上报、命令下发等。旨在通过物联网检测技术帮助开发者发现自身产品中存在的缺陷或问题，缩短产品上市时间。测试完成后，开发中心将生成测试报告，用于进行产品发布认证。当开发者已完成产品profile定义和编解码插件开发，并且部署了编解码插件后，可以进行自助测试。

**步骤1** 在产品开发空间，单击“发起自助测试”。

**步骤2** 进入“选择测试用例”界面，开发者可以自行选择需要测试的用例，系统会自动检查已选测试用例是否满足测试能力并返回检查结果。

- 如果所有已选测试用例检查通过，则单击“下一步”继续下一阶段测试。
- 如果有测试用例检查未通过，则单击该测试用例的“信息缺失”，根据提示信息进行修改。

**注：**产品通过测试的用例越多，产品发布到产品中心的审核通过率就越高。建议软/固件升级测试任选其一，其余测试用例全选。



**步骤3** 根据测试用例说明，依次完成自助测试。完成测试后，可以预览测试报告或申请发布产品。

---结束

## 设备绑定测试

设备绑定测试用于测试设备接入物联网平台的能力，包括设备注册和设备上线两个步骤。

### 📖 说明

设备绑定测试是进行其他测试的前提条件，如果测试失败，则无法进行其他测试。

**步骤1** 在设备绑定测试界面，单击“下一步”，进入注册设备界面。

**步骤2** 根据向导进入测试界面，选择“安全模式”，并填写“设备标识”和“模组名称”，单击“下一步”。

- 如果安全模式选择“加密”，还需要填写“PSK”。
- 如果不是使用模组进行测试，则“模组名称”填写无。

**步骤3** 根据向导操作真实设备接入物联网平台进行绑定，查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。

---结束

## 数据上报测试

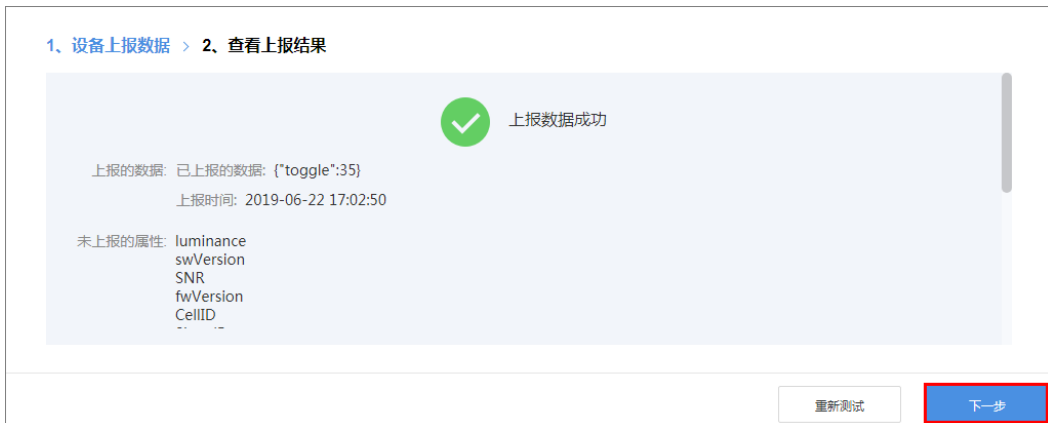
数据上报测试用于测试设备上报数据的能力，目的是验证Profile中定义的属性字段是否正确。如果物联网平台与设备交互的“数据格式”为二进制码流，还会验证编解码插件与Profile的映射关系是否正确。

**步骤1** 在数据上报测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，操作真实设备上报Profile中定义的属性数据，如果设备的所有属性数据都已上报完成，可以单击“终止测试”完成数据上报测试并查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。

平台会检测所有上报成功的属性数据，并记录到测试报告中，重复上报的属性只会记录一次。



----结束

## 无线参数上报测试

无线参数上报测试用于测试设备上报无线参数属性数据的能力，包括信号强度、覆盖等级、信噪比和小区ID。

如果测试此用例，需要在Profile中定义如下无线参数属性，并在编解码插件中建立对应的映射关系。

参数	类型	描述
RSRP/rsrp/ signalStrength/ SignalPower	int	信号强度，取值范围-140~-40。
ECL/signalECL	int	覆盖等级，取值范围0~2。
SNR/snr/SINR/sinr/ signalSNR	int	信噪比，取值范围-20~30。
CellID/cellId	int	小区ID，取值范围0~2147483647。

**步骤1** 在无线参数上报测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，操作真实设备上报Profile中定义的无线参数属性数据，查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。

#### 📖 说明

上报的无线参数值必须在数据范围内才算有效数据。



----结束

## 控制命令下发测试

控制命令下发测试用于测试设备接收和处理控制命令的能力，目的是验证Profile中定义的命令字段是否正确。如果平台与设备交互的“数据格式”为二进制码流，还会验证编解码插件与Profile的映射关系是否正确。由于下发的命令是立即下发模式，设备需要在线。

如果使用业务应用进行测试，还会测试业务应用是否正确调用物联网平台“创建设备命令”接口给设备下发命令的能力。

**步骤1** 在控制命令下发测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，物联网平台会根据Profile的定义向设备下发一条命令。在真实设备应答后，查看测试用例执行结果。

如果是使用业务应用接入物联网平台，则操作业务应用向设备下发一条命令，在真实设备应答后，进入“上传应用下发命令截图”界面，单击界面中的“+”，上传业务应用命令下发成功的截图。此图作为业务应用正确调用物理网平台“创建设备命令”接口的凭证。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。



----结束

## 命令下发响应测试

命令下发响应测试用于测试设备在接收物联网平台命令后主动上报执行结果的能力。当Profile中定义了“命令下发响应字段”，即设备需要返回命令执行结果时，开发者才需要进行命令下发响应测试。由于下发的命令是立即下发模式，设备需要在线。

**步骤1** 在命令下发响应测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，物联网平台会根据Profile的定义向设备下发一条命令，如果是业务应用接入物联网平台，则操作业务应用向设备下发一条命令。

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，直接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，手动操作真实设备上报命令执行结果后，查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。



----结束

## 固件升级测试

固件升级测试用于测试设备是否具备固件升级能力。在执行固件升级测试前，请确认“Profile定义 > OM维护”下，“固件升级”已开启。

**步骤1** 在固件升级测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，上传未签名的固件升级包，填写版本号，单击“下一步”，系统自动创建固件升级任务。

请确保上传的升级包用于该设备的固件升级，且文件为zip格式压缩包。



**步骤3** 操作真实设备上报一条属性数据，触发升级任务，待升级任务执行完成后，查看升级结果。

- 升级成功，单击“下一步”验证设备升级后是否正常工作。
- 升级失败，排查并处理问题后，单击“重新测试”重测固件升级。



**步骤4** 操作真实设备上报一条属性数据，验证设备升级后是否能与物联网平台正常通信，查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。





----结束

## 软件升级测试

软件升级测试用于测试设备是否具备软件升级能力。在执行软件升级测试前，请确认“Profile定义 > OM维护”下，“软件升级”已开启。

**步骤1** 在软件升级测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，上传未签名的软件升级包，单击“下一步”，系统自动创建软件升级任务。

请确保上传的升级包用于该设备的软件升级，且文件为zip格式压缩包。



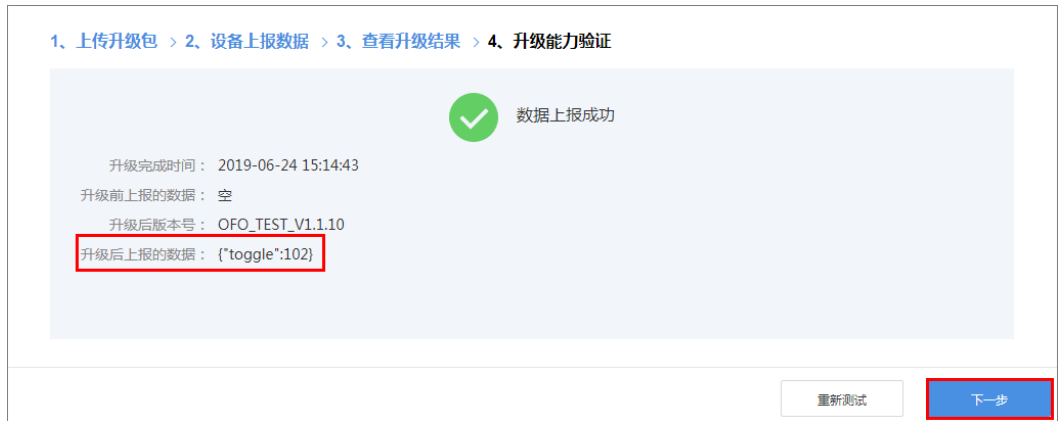
**步骤3** 操作真实设备上报一条属性数据，触发升级任务，查看测试用例执行结果。

- 升级成功，单击“下一步”验证设备升级后是否正常工作。
- 升级失败，排查并处理问题后，单击“重新测试”重测软件升级。



**步骤4** 操作真实设备上报一条属性数据，验证设备升级后是否能与物联网平台正常通信，查看测试用例执行结果。

- 测试成功，单击“下一步”进入下一阶段测试。
- 测试失败，排查并处理问题后，单击“重新测试”重测测试用例。



----结束

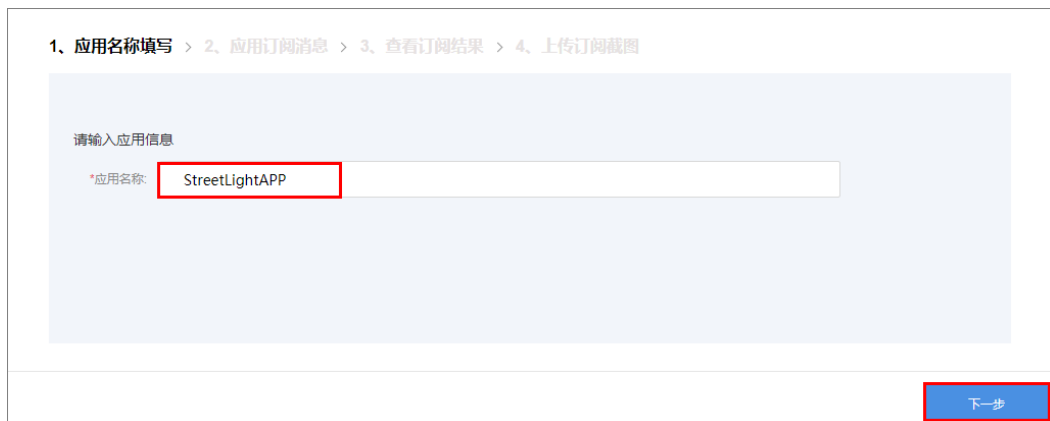
## 应用订阅事件测试

应用订阅事件测试主要是测试业务应用是否可以正确调用物联网平台“订阅平台业务数据”接口，订阅设备数据变化通知的能力。

如果物联网平台使用HTTPS协议向业务应用推送数据时，需要在物联网平台上传业务应用提供的CA证书。可以在开发中心的“应用 > 对接信息 > 推送证书”，单击“证书管理”上传证书，证书获取参考[如何导出https推送证书](#)。

**步骤1** 在应用订阅事件测试界面，单击“下一步”开始测试。

**步骤2** 在“应用名称填写”界面，填写应用名称，单击“下一步”。



**步骤3** 根据向导进入测试界面，在业务应用订阅设备数据变化消息，查看测试用例执行结果。

- 应用消息订阅成功，单击“下一步”进入“上传订阅截图”界面。
- 应用消息订阅失败，在业务应用排查并处理订阅故障后，单击“重新测试”重测测试用例。

**步骤4** 单击“上传订阅截图”界面中的“+”，上传业务应用订阅成功的截图。此图是业务应用正确调用物联网平台“订阅平台业务数据”接口的凭证。

图片上传成功后，单击“下一步”进入下一阶段测试。

#### 📖 说明

上传的图片大小不能超过20MB。

----结束

## 应用接收推送数据测试

应用接收推送数据测试主要是测试业务应用是否可以正确接收物联网平台推送设备数据的能力。

**步骤1** 在应用接收推送数据测试界面，单击“下一步”开始测试。

**步骤2** 根据向导进入测试界面，操作真实设备上报一条profile中定义的属性数据，物联网平台获取数据并推送给业务应用，查看测试用例执行结果。

- 数据推送成功，单击“下一步”进入“上传应用接收截图”界面。
- 数据推送失败，排查并处理故障后，单击“重新测试”重测测试用例。

**步骤3** 单击“上传应用接收截图”界面中的“+”，上传业务应用接收到物联网平台推送数据的截图。此图是业务应用正确接收物联网平台推送数据的凭证。

#### 📖 说明

上传的图片大小不能超过20MB。

----结束

# 7 产品发布（联通用户专用）

非联通用户请查看[设备接入服务](#)。

如果开发中心已经对接产品中心，则开发者在完成产品的[自助测试](#)后，可以把产品发布到产品中心，已发布的产品可直接应用于商用环境。

## 申请发布产品

**步骤1** 产品在通过所有测试用例后，单击“申请发布”。

**步骤2** 系统自动完成厂商信息和产品信息完整性的检查。如果没有重要信息缺失，则单击“发布”。

- 黄色信息缺失提示：部分信息不完整，不影响发布产品，但发布到产品中心可能会审核不通过，建议补充。
- 红色信息缺失提示：重要信息缺失，需补充完整才能发布产品。

产品信息				发布
厂家LOGO检查			修改	
厂家名称检查	HW		修改	
厂家联系方式	12345678901		修改	
厂家简介检查	提供更好的物联网产品及服务。		修改	
产品介绍	信息缺失	补充完整之后可点击“发布”按钮，将产品发布至产品中心	修改	
功能亮点	信息缺失	发布至产品中心可能会审核不通过，建议补充	修改	
产品规格	智慧路灯		修改	
服务支撑	信息缺失	发布至产品中心可能会审核不通过，建议补充	修改	
客户案例	信息缺失	发布至产品中心可能会审核不通过，建议补充	修改	
产品图片			修改	

**步骤3** 选择发布方式：“公开发布”或“私有发布”，单击“发布”，提交发布申请。

**注：**一旦发布不可修改发布方式。



**步骤4** 查看产品是否发布到产品中心。

登录[认证产品中心](#)，选择“产品”，在产品列表查看通过审核的产品。

**步骤5**（可选）申请关联云市场产品，以使用户通过产品中心直接购买该产品。

1. 进入认证产品中心，鼠标移至右上角的用户名，单击下拉列表中的“我的产品”。



2. 在产品列表，单击产品右侧的“云市场发布”。



3. 系统弹出“云市场发布”窗口，单击“填入商品ID”，然后单击“确定”。  
**注：**“商品ID”为云市场产品的产品ID，在产品审核上架后，可在云市场中该产品的商品详情页面获取。云市场发布产品请参考[云市场商品接入](#)。

## 云市场发布



填入商品ID之后，产品中心的商品即可出售，若您还没上架，请先去云市场完成相关操作

- 前往云市场申请发布商品
- 等待华为审核发布申请
- 审核通过，填入商品ID

确定

取消

----结束

# 8 商用对接（联通用户专用）

非联通用户请查看[设备接入服务](#)。

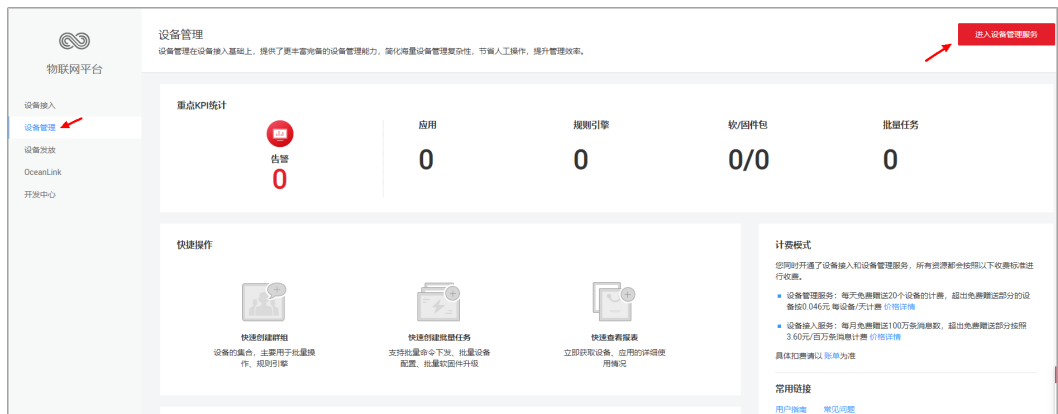
当产品发布到产品中心后，我们在“开发中心”定义好的产品就可以在“设备管理服务”中使用了。由于开发中心所在的平台环境为测试环境，设备管理服务所在的平台环境为商用环境，两个环境的数据不能互通，已对接测试环境的设备/应用服务器要迁移商用环境，您需要通过本章节的指引完成商用对接。

## 总体说明

- 迁移对接的过程中，开发中心数据不割接，由应用服务器负责保存设备在开发中心的历史数据。
- 建议应用服务器先迁移商用环境再迁移设备，设备迁移采用API逐个重新注册的方式。

## 创建应用

**步骤1** 选择“设备管理”，登录设备管理服务控制台。



**步骤2** 系统已自动为用户创建一个应用，若使用默认应用，需要在“应用定义 > 安全”中重置密钥，且无需执行**步骤3**至**步骤4**。

如果用户想要自己创建应用，需要将默认应用删除，重新创建。

应用管理 > 创建应用

---

**基本信息**

\*应用名称:   
名称只能包含大写字母、小写字母、数字和下划线,且不能大于50个字符

企业:

所属行业:

\*消息跟踪数据授权:  打开  关闭  
关闭授权开关,平台管理员在辅助SP用户进行设备的故障定位时,不能跟踪设备上报的业务数据,可能导致没有足够的信息,将会降低问题定位效率,建议您授权给平台管理员进行业务数据的跟踪。

---

**消息推送**

**选择协议** 平台支持应用服务器通过HTTP和HTTPS两种协议接入方式,推荐采用HTTPS协议,安全性更高

推送协议:

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

CA证书 :


**步骤3**（使用默认应用无需执行此步骤）参考下表按照实际情况填写配置参数。

参数名称	参数说明
<b>基本信息</b>	
应用名称	定义用户的应用的名称,应用名称必须为帐号下唯一,且创建后不可更改。
所属行业	根据用户的应用的行业属性进行选择。
消息跟踪数据授权	<p>设置物联网平台运营管理员可以跟踪发生故障的设备的权限。</p> <ul style="list-style-type: none"> <li>打开授权,表示平台管理员在辅助租户进行设备的故障定位时,可以跟踪设备上报的业务数据,便于快速解决问题。授权打开的状态下需要设置“授权时效”,可设置“指定时间”或者“永久有效”。为了保证用户的数据权益,物联网平台运维管理员跟踪的设备数据保留时间不超过3天。</li> <li>关闭授权,表示平台管理员在辅助租户进行设备的故障定位时,不能跟踪设备上报的业务数据,可能导致没有足够的信息,将会降低问题定位效率,建议您授权给平台管理员进行业务数据的跟踪。</li> </ul>
<b>消息推送</b>	



参数名称	参数说明
选择协议	<p><b>推送协议</b></p> <p>应用服务器向物联网平台进行消息订阅，物联网平台在推送数据时可以采用加密的HTTPS协议或者非加密的HTTP协议。推荐采用HTTPS协议。</p> <ul style="list-style-type: none"> <li>• HTTPS方式：表示物联网平台与应用服务器之间采用加密的传输协议，需要应用服务器侧上传CA证书。</li> <li>• HTTP方式：表示物联网平台与应用服务器之间采用非加密的传输协议。此方式的安全性较低，存在物联网平台与应用服务器之间通信信息泄露风险。</li> </ul> <p><b>CA证书</b></p> <p>创建应用时无需上传CA证书，请创建完应用后根据<a href="#">配置数据推送业务</a>上传CA证书。</p>
<b>平台能力</b>	
设备数据处理	<p>物联网平台提供设备上报数据的存储能力，用户可以通过“存储历史数据”的开关进行控制，默认为“打开”状态。</p> <ul style="list-style-type: none"> <li>• 打开开关：即物联网平台会对上报数据进行存储，存储时间以界面显示的存储时间为准。</li> <li>• 关闭开关：即物联网平台不对上报数据进行存储。</li> </ul> <p>用户可以通过<a href="#">数据转发规则</a>转发到华为云其他云服务上进行存储和处理。</p>
推送服务	应用服务器向物联网平台订阅设备信息，物联网平台能够向应用服务器进行消息推送。
<b>其他</b>	
应用描述	对该应用的描述。
应用图标	为该应用添加自定义图标。

**步骤4**（使用默认应用无需执行此步骤）勾选“我已阅读并同意《个人数据使用条款》”，单击“确定”，完成创建应用。创建完成后，系统弹出“成功”对话框，显示应用的基本信息，包含应用ID、应用密钥、应用对接地址和设备对接地址。

- 请单击“保存密钥至本地”，以保存应用密钥信息，密钥信息在应用详情页内不可见，请妥善保管。如果遗忘应用密钥时，可在“应用列表”中单击，选择“重置密钥”，或者通过应用详情页内“应用定义 > 安全”进行重置密钥。

#### 说明


应用ID和应用密钥用于应用服务器接入物联网平台，如果重置密钥，旧的密钥将不能使用，您的应用服务器需要更新为新的密钥才能重新接入平台，请谨慎操作。

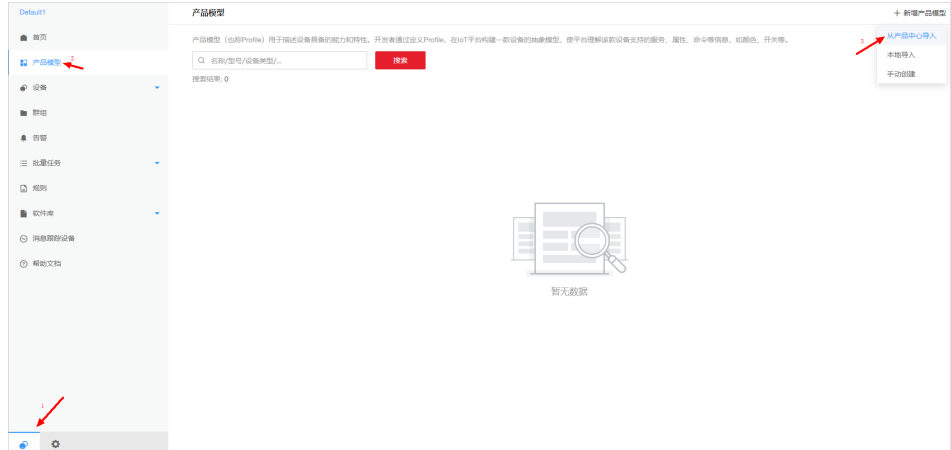
- 单击“查看应用详情”进入应用详情页，具体功能介绍参照“应用详情”。
- 单击“返回应用列表”，返回到创建应用页。单击“应用列表”中应用的图标，可直接进入该应用的应用详情页。

----结束

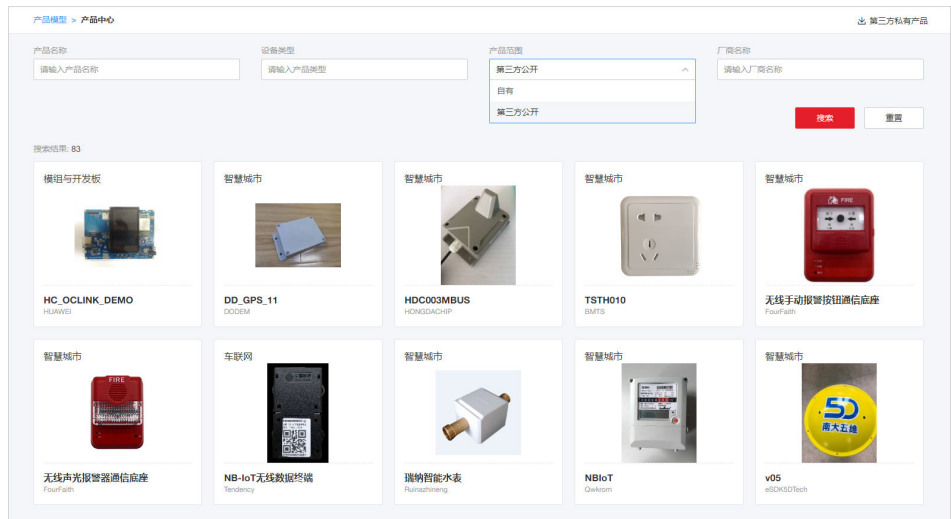
## 导入产品模型

- 从产品中心导入产品模型

- 单击页面左下角的  切换左侧菜单，打开“产品模型”页面，点击页面右上角的“新增产品模型 > 从产品中心导入”。




- 在产品中心选择已发布的产品，导入即可。

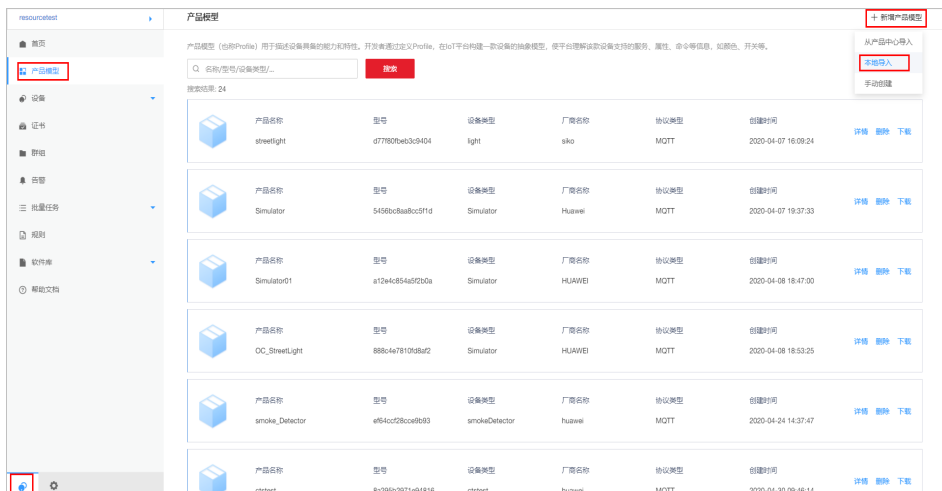


### 说明

如果您需要导入第三方私有产品，单击右上角“第三方私有产品”，填写产品验证码。产品验证码由第三方私有产品提供方在认证产品中心的“我的产品”页面，单击该产品的“获取产品码”得到。

- 本地导入产品模型

- 单击页面左下角的  切换左侧菜单，打开“产品模型”页面，点击页面右上角的“新增产品模型 > 本地导入”



- b. 在弹出的页面，填写“产品名称”，导入本地产品模型的资源文件，点击“确认”按钮。



### 说明

选择“本地导入”的方式导入产品模型，如果产品模型的协议类型为LWM2M/CoAP，则需要联系华为技术支持上传插件。推荐您将产品发布到产品中心后再选择“从产品中心导入产品模型”。

## 迁移应用

迁移应用是指将对接了开发中心（测试环境）的应用服务器迁移到设备管理服务（商用环境）中去，您需要完成以下步骤：

- 步骤1** 为了使应用服务器能对接这个业务使用空间，您需要将对接的平台地址修改为“设备管理服务”的应用对接地址，应用对接地址获取请参考[平台对接信息](#)。
- 步骤2** 在“设备管理服务”中，应用对应的是开发中心的项目。为了使应用服务器能调用平台的接口，您需要将应用服务器中设定的应用ID和应用密钥修改为新的值。

**步骤3** 如果应用服务器调用“鉴权”接口成功，表明应用服务器已对接到商用环境。

----结束

## 迁移设备

迁移设备是指将对接了开发中心（测试环境）的设备迁移到设备管理服务（商用环境）中去。迁移对接的过程中，开发中心数据不割接，由应用服务器负责保存设备在开发中心的历史数据。

**步骤1** 您需要将设备根据通信协议修改对接的平台地址为“设备管理服务”的设备对接地址，设备对接地址获取请参考[平台对接信息](#)。

**步骤2** 您需要在“设备管理服务”中注册设备，可以使用应用服务器调用平台注册设备接口，也可以在界面上注册设备。

**步骤3** 平台为每一个注册到平台上的设备生成了一个deviceId，这是设备在平台上的唯一ID，您需要将设备和ID映射关系刷新为新的ID。

----结束

## 对接验证

将真实的设备和应用接入到物联网平台后，需要对设备上报数据到物联网平台，平台能正常将设备上报的数据推送给应用服务器进行验证；同时，验证应用服务器向设备下发命令，设备能正常收到命令并执行成功。

**步骤1** 设备上电，基于在设备上定义的业务逻辑进行数据采集，向物联网平台上报数据。

**步骤2** 登录“设备管理服务控制台”，选择“设备管理 > 设备 > 所有设备”，在设备列表中查看对应设备的状态。如果状态为“在线”，则表示设备已经成功接入物联网平台。



**步骤3** 点击对应的设备，进入设备详情页，在详情页中查看“最近上报数据”，如果能正常解析和显示对应的数据，则表示设备上报数据成功。

如果需要查看所有上报的历史数据，则可以在设备详情的“历史数据”中进行查看。



**步骤4** 在CallbackURL对应的服务端中，查看是否收到物联网平台推送的数据，如果能正常接收，则表示物联网平台推送消息成功。

**步骤5** 通过应用服务器向设备下发命令，在设备侧查看设备的执行结果，如果设备的执行动作与下发的命令相符，且在“设备管理服务控制台”中查看下发命令任务的执行结果为“已送达”或“成功”，则表示应用服务器向设备下发命令成功。

- 对于NB-IoT设备，如果采用的是缓存下发模式，需要触发设备再次上报数据后，命令才会下发给设备。
- 如果设备会给物联网平台返回命令的执行结果（成功或失败），则命令下发的任务状态会根据执行结果刷新为“成功”或“失败”。

---结束

## 开始使用

物联网平台设备提供海量设备的接入和管理能力，配合华为云其他产品同时使用，帮助快速构筑物联网应用，简化海量设备管理复杂性，节省人工操作，提升管理效率。

功能	简介
<b>应用管理</b>	应用可以理解为在物联网平台中为用户的业务划分一个项目空间，当用户在开发中心、线下环境里完成应用服务器侧的开发、物联网平台侧的开发以及设备侧的开发后，就需要在控制台上创建应用，将开发完的应用服务器与真实设备接入到这个项目空间中，实现设备的数据采集和设备管理。
<b>产品模型</b>	又称Profile，用于定义一款接入设备所具备的属性（如颜色、大小、采集的数据、可识别的指令或者设备上报的事件等信息），然后通过厂家、设备类型和设备型号，唯一标识一款设备，便于平台识别。产品模型可通过开发中心进行无码化开发。
<b>设备注册鉴权</b>	物联网平台对接入平台的设备进行鉴权认证。待真实设备上电后，设备可以上报数据到物联网平台，物联网平台根据应用服务器的订阅消息类型，把消息推送给应用服务器。
<b>订阅推送</b>	订阅：是指应用服务器通过调用物联网平台的API接口，向平台获取发生变更的设备业务信息（如设备注册、设备数据上报、设备状态等）和管理信息（软固件升级状态和升级结果）。 推送：是指订阅成功后，物联网平台根据应用服务器订阅的数据类型，将对应的变更信息推送给指定的URL地址。
<b>数据上报</b>	当设备完成和物联网平台对接后，一旦设备上电，设备基于在设备定义上的业务逻辑进行数据采集和上报，可以是基于周期或者事件触发。
<b>命令下发</b>	为能有效地对设备进行管理，设备的产品模型中定义了物联网平台可向设备下发的命令，应用服务器可以调用物联网平台开放的API接口向单个设备或批量设备下发命令，或者用户通过物联网平台直接向单个设备下发命令，配置或修改设备的服务属性值，以实现设备的远程控制。
<b>设备配置更新</b>	物联网平台提供设备配置更新功能，即用户可通过控制台对单个设备或批量设备的设备属性值进行修改，满足用户频繁、快捷、方便的管理设备的诉求。
<b>设备影子</b>	设备影子是一个JSON文件，用于存储设备的在线状态、设备最近一次上报的设备属性、应用服务器期望下发的配置。每个设备有且只有一个设备影子，设备可以获取和设置设备影子以此来同步状态，这个同步可以是影子同步给设备，也可以是设备同步给影子。

功能	简介
<b>规则引擎</b>	指用户可以在物联网平台上可以对接入平台的设备设定相应的规则，在条件满足所设定的规则后，平台会触发相应的动作来满足用户需求。包含设备联动和数据转发两种类型。
<b>群组与标签</b>	<p>群组是一系列设备的集合，用户可以对应用下所有设备，根据区域、类型等不同规则进行分类建立群组，以便处理对海量设备的批量管理和操作。</p> <p>物联网平台支持定义不同的标签，并对设备打标签，通过标签，可以快速筛选设备。</p>
<b>设备监控</b>	提供查看设备详情、设备状态管理、查看报表、查看操作记录、查看审计日志、告警管理、设备消息跟踪等设备监控与运维能力，提升设备的可维护性。
<b>远程诊断</b>	支持用户对接入的设备进行远程维护操作，快速定位问题及恢复业务，减少近端维护引入的高成本。当前支持的远程维护操作包括设备的运行日志收集、重启模组。
<b>固件升级</b>	用户可以通过OTA的方式对支持LWM2M协议的设备进行固件升级，升级协议为LWM2M协议。
<b>软件升级</b>	用户可以通过OTA的方式支持对LWM2M协议的设备进行软件升级，升级协议为PCP协议。
<b>网关与子设备</b>	物联网平台支持设备直连，也支持设备挂载在网关上，作为网关的子设备，由网关直连，通过网关进行数据转发。

# 9 IoT 技术认证（联通用户专用）

非联通用户请查看[设备接入服务](#)。

## 认证概述

华为IoT技术认证是华为面向合作伙伴提供的IoT技术认证服务，旨在通过严谨而专业的检测技术帮助合作伙伴发现并解决自身产品的不足，缩短产品上市周期，同时联合合作伙伴向共同的客户提供经过验证的方案。通过该认证的伙伴产品将获得华为颁发的IoT技术认证证书，该证书标志着华为对双方物联网产品预集成的技术认可。

技术认证特点：

- 权威性：提供华为商业化平台认证服务、国家/区域重要法规认证服务、重要行业安全标准认证服务，得到许多行业客户以及运营商的认可。
- 全面性：完备的测试环境和专业的测试服务，从功能、性能、可靠性、安全、可维护性等为物联网产品提供端到端的软硬件能力测试，高效打造优质产品。
- 多渠道推广：达到一定标准的产品可以在产品中心发布，获得证书的产品不仅可以在产品中心、华为云市场发布，而且还可以享受华为提供的商业推广机会，例如展会活动、联合营销等。
- 方便快捷：OpenLab认证实验室全球覆盖、线上一站式申请、线下专业人员指导，为全球合作伙伴提供贴身的技术认证服务。

### 说明

华为会针对行业特点对认证产品的安全进行必要的验证，但不承担合作伙伴所属产品的安全责任。

## 认证分类

根据产品的认证方式、认证测试内容和认证目的不同，分为Enabled和Compatible两种认证证书。

		
<b>认证方式</b>	线上自助测试	OpenLab现场测试
<b>认证测试内容</b>	验证业务应用和设备能否正确接入物联网平台，并且实现基础功能，如数据上报、命令下发、应用订阅事件等。	全面验证业务应用和设备能否可靠安全的接入物联网平台，功能、性能、可靠性、安全、可维护性等能达到华为物联网平台标准。
<b>认证目的</b>	指定的产品是基于华为开放的产品能力和技术支持服务开发的，经认证测试有效调用了华为产品的ICT能力	指定的产品同华为产品完成了互联互通测试，能够可靠安全地接入华为物联网平台，且满足协议规定的性能要求。

因为设备的通信技术和集成方式不同，所以将Compatible证书细化为NB-IoT Compatible、Agent Lite Compatible和Agent Tiny Compatible三种认证类型。

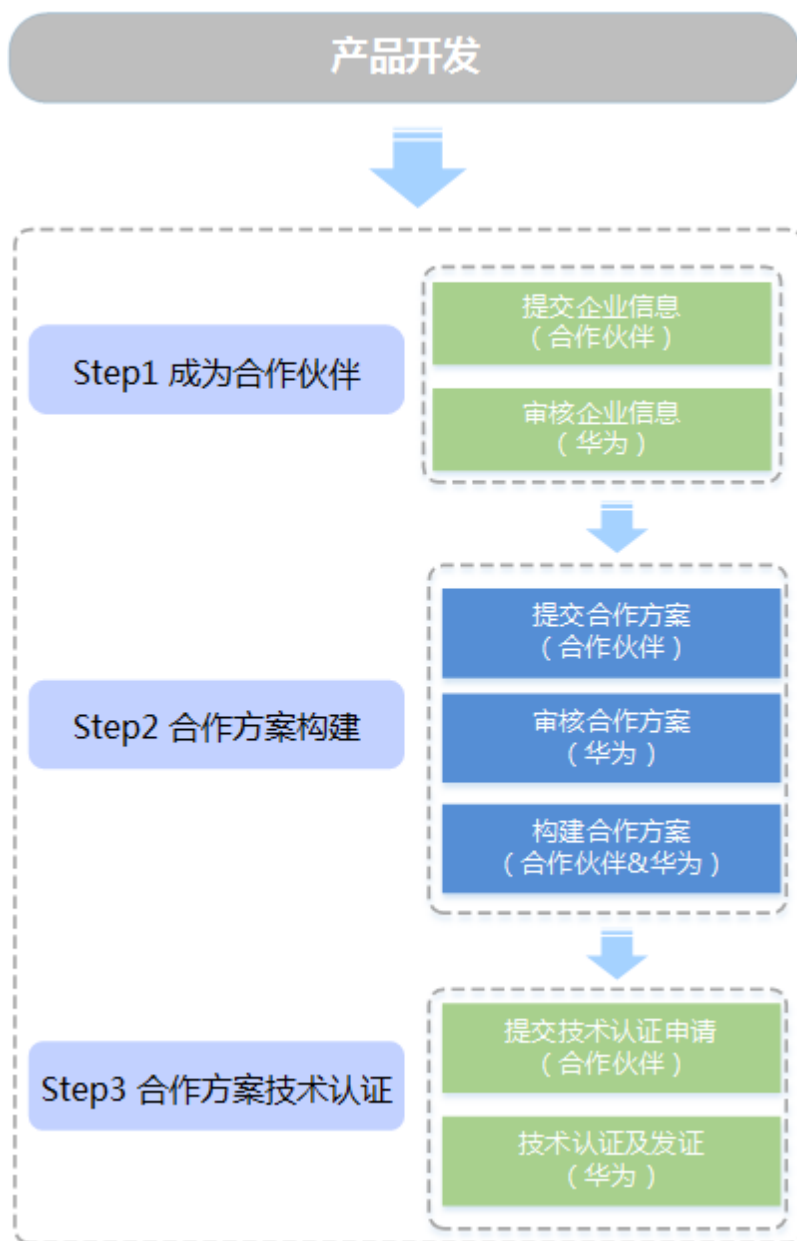
认证证书	认证类型	认证对象	认证测试指导
Enabled	Enabled	业务应用	测试指导请参考 <a href="#">自助测试（联通用户专用）</a> 。
Compatible	NB-IoT Compatible	使用NB-IoT通信技术的设备	测试指导请参考 <a href="#">OpenLab现场测试</a> 。
	Agent Lite Compatible	集成Agent Lite SDK的设备	
	Agent Tiny Compatible	集成LiteOS SDK的设备	

如需申请技术认证请联系oceanconnect@huawei.com。

## OpenLab 现场测试

合作伙伴在物联网平台上完成产品开发（包括[平台侧开发](#)、[设备侧开发](#)和[应用侧开发](#)）后，然后才可以进行OpenLab技术认证，认证流程如下。





关键操作	说明	相关资源
成为合作伙伴	提交企业相关信息，成为合作伙伴，可以申请合作伙伴的权益。 <b>注：</b> 如果未注册华为账号，需先进行华为账号注册。	<a href="#">成为合作伙伴</a>
合作方案构建	合作伙伴根据华为提供的构建指导书，提交合作方案资料。通过专业检测服务完成该方案的构建和优化。	-

关键操作	说明	相关资源
合作方案 技术认证	合作方案完成构建后，合作伙伴可以申请华为技术认证。通过认证的方案，将获得华为颁发的技术认证证书，同时该技术认证结果将公布到华为技术认证证书网站供合作伙伴和客户查询。	<ul style="list-style-type: none"><li>● <a href="#">华为IoT应用认证指导 (Enabled)</a></li><li>● <a href="#">华为NB-IoT终端认证指导(Compatible)</a></li><li>● <a href="#">华为IoT Agent Lite终端认证指导 (Compatible)</a></li><li>● <a href="#">华为IoT Agent Tiny终端认证指导 (Compatible)</a></li></ul>