设备管理

开发指南

文档版本	02
发布日期	2019-08-28

华为技术有限公司

商标声明


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

注意

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

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

目 录

1 产品开发

1.1 开发资源获取

应用开发资源

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和SDK包。应用开发即应用侧通过调用物联网平台的API，实现安全接入、设备管理、数据采集、命令下发等业务场景的过程。请根据需要下载对应的资源文件。

资源包名	描述	下载
应用开发北向 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北向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**IoT平台北向API参考**和**应用开发指导**。	**应用开发北向 API JAVA Demo**

资源包名	描述	下载
应用开发北向 Java SDK	Java SDK提供JAVA方法调用物联网平台北向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**IoT平台北向JAVA SDK API参考**和**JAVA SDK 使用指南**。	● **JAVA SDK** ● **JAVA SDK Demo**

设备开发资源

IoT平台支持设备通过MQTT协议和LWM2M/CoAP协议进行接入，设备可以通过调用南向的设备接口或者集成SDK的方式接入到IoT平台。

资源包名	描述	下载
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。使用指导可以参考**LiteOS SDK端云互通组件开发指南**。	**LiteOS SDK**
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**Profile文件线下开发参考**。	**Profile开发示例**
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**编解码插件线下开发参考**。	**编解码插件开发样例**
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。	**编解码插件检测工具**
NB-IoT设备模拟器	用于模拟以CoAP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。	**NB-IoT设备模拟器**

证书获取

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

此证书包只用于与华为公有云IoT平台的对接。

证书包的目录结构和各证书的用途详见表1-1。

表 1-1 证书介绍

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

1.2 创建项目和产品

概述

- 项目：创建一个项目，相当于为用户分配一个独立的项目空间，开发者可以在项目空间中开发相应的物联网产品和应用。
- 产品：某一类具有相同能力或特征的设备的集合称为一款产品。除了设备实体，产品还包含该类设备在物联网能力建设中产生的产品信息、产品模型（Profile）、插件、测试报告等资源。

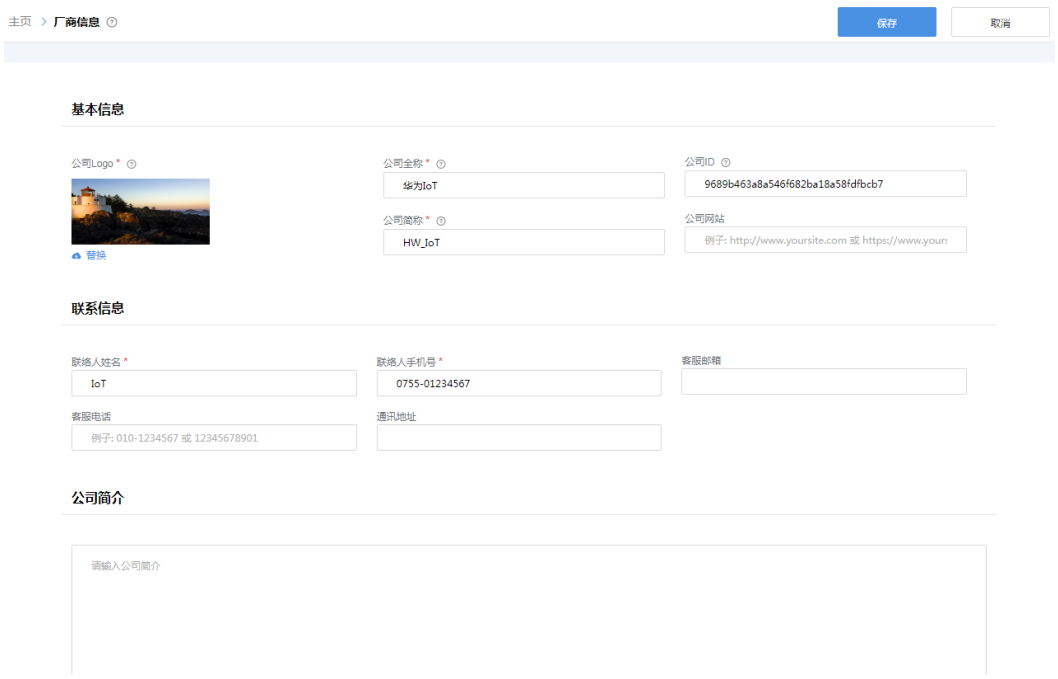
操作指导

步骤1 访问IoT设备管理服务首页，点击进入“开发中心”。

说明

目前开发中心仅开放香港站点，请在香港站点的开发中心完成产品开发后，再到对应的站点进行设备和应用的对接。

步骤2 （可选）如果用户是第一次使用开发中心，请在开发中心首页，选择右上角的“厂商信息”，根据公司实际情况完善厂商信息，点击“保存”。



步骤3 在开发中心首页，点击“新建项目”，然后填写项目名称和选择所属行业后，点击“确定”。

步骤4 创建项目完成后，会弹出“项目创建成功”提示框，请将应用ID和应用密钥下载到本地进行保存，然后点击“进入项目”。

说明

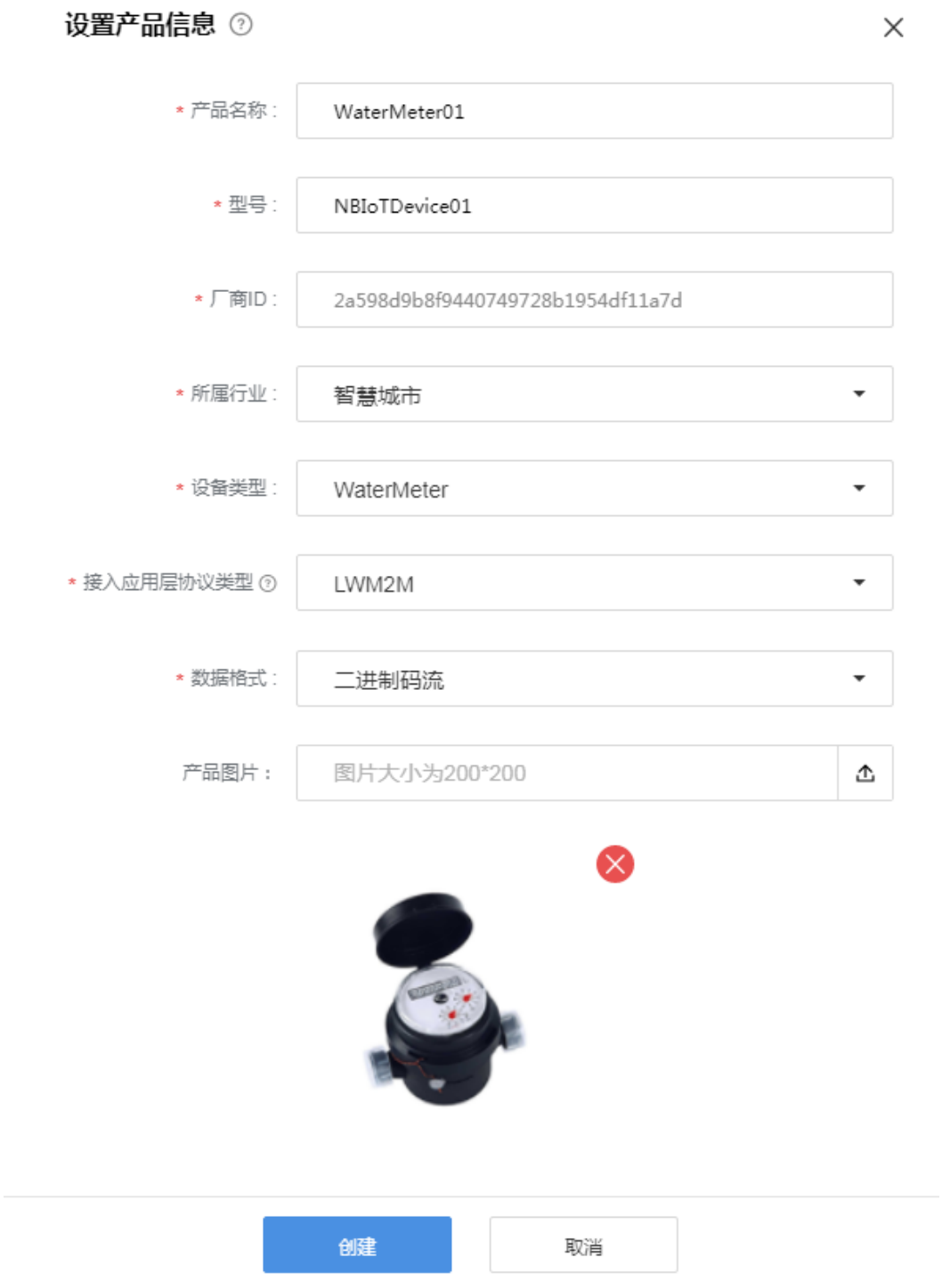
应用ID和应用密钥在应用对接物联网平台时需要这两个参数，请妥善保存，如果遗忘，可以在该项目的“应用 > 对接信息 > 应用安全”中进行重置。

步骤5 进入项目空间首页后，在“产品 > 产品开发”界面中，点击“新建产品”。

步骤6 创建新产品有两种方式，一是基于系统预置的模板进行创建，另一种是自定义一款新产品。本示例已“自定义产品”为例进行说明。

步骤7 在自定义产品页签中，点击“自定义产品”。

步骤8 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“所属行业”等信息后，点击“创建”。



步骤9 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

1.3 开发产品模型

1.3.1 开发指导

概述

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

- **设备能力 (Device Capability)**

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

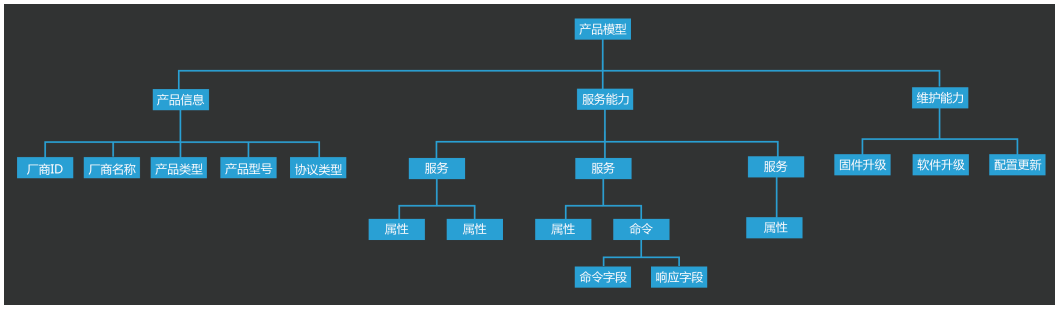
  例如：水表的制造厂商为“HZYB”，制造商ID为“TestUtf8ManuId”，型号为“NBIoTDevice”，协议类型为“CoAP”。

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

- **服务(Service)**

  描述设备具备的服务能力，每个服务具备的属性、命令以及命令的参数。

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

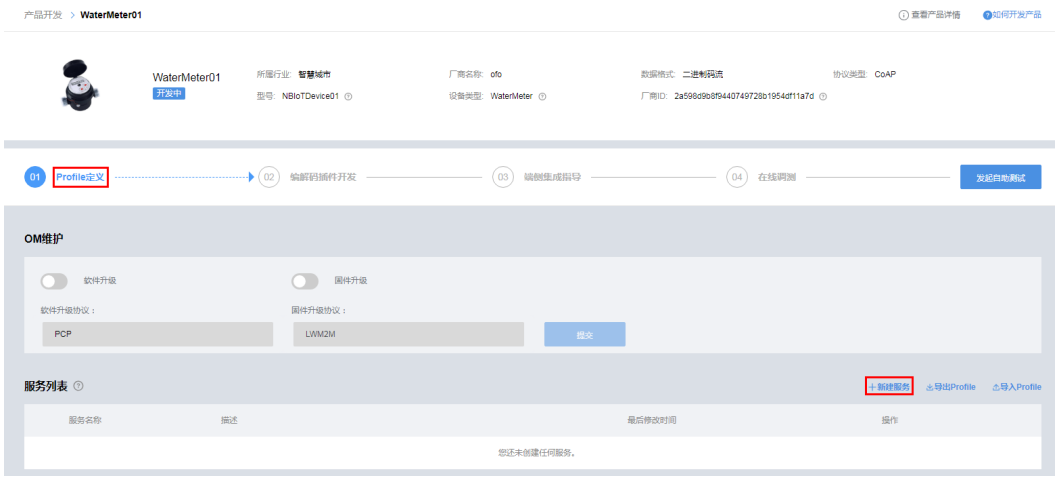


操作指导

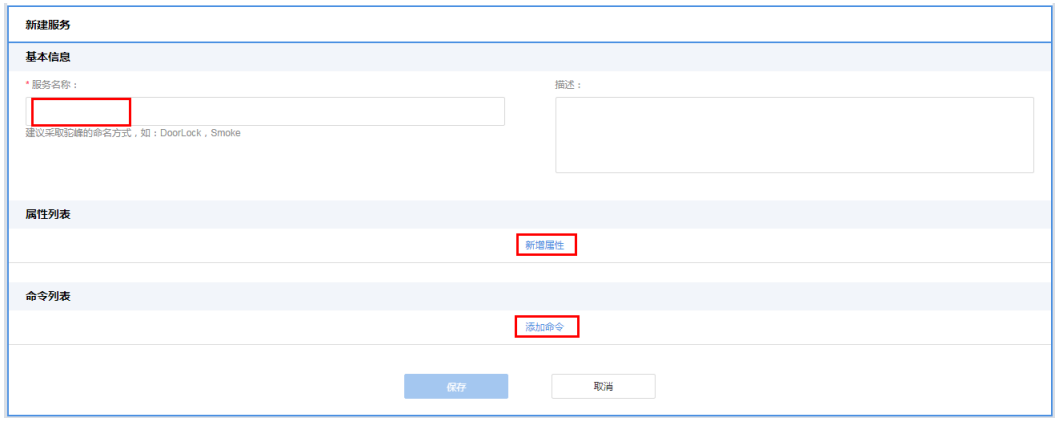
在创建项目和产品时：如果选择使用系统模板，则系统将会自动使用相应的Profile模板，开发者可以直接使用或在此基础上进行修改；如果选择自定义产品模板，则需要完整定义Profile，操作如下：

步骤1 在“产品开发”界面选择产品，选择具体产品，进入该产品的开发空间。

步骤2 在产品开发空间，点击“Profile定义”，然后点击“新建服务”。



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

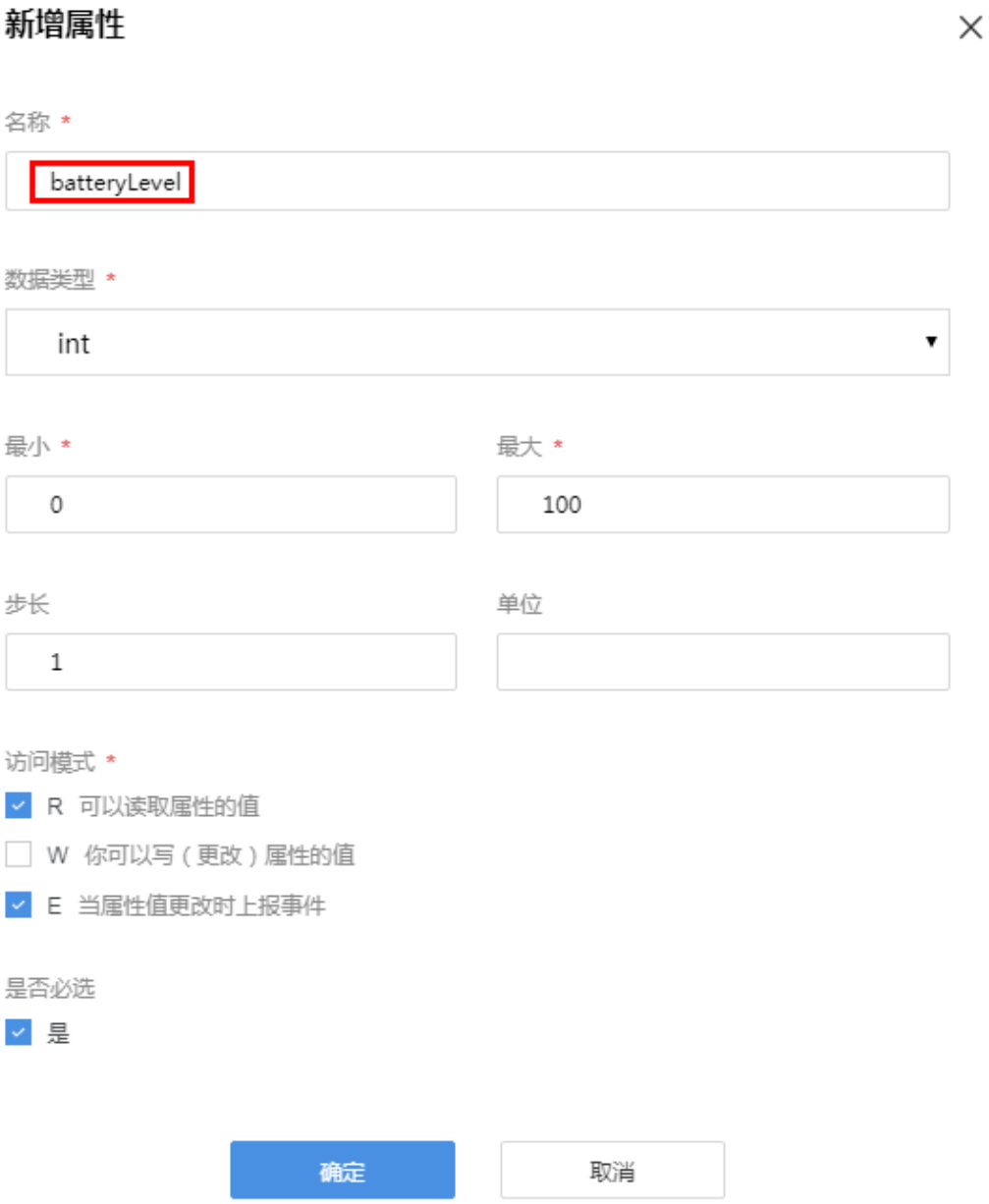


1. 填写“服务名称”，“服务名称”采用首字母大写的命名方式，比如：WaterMeter、Battery。

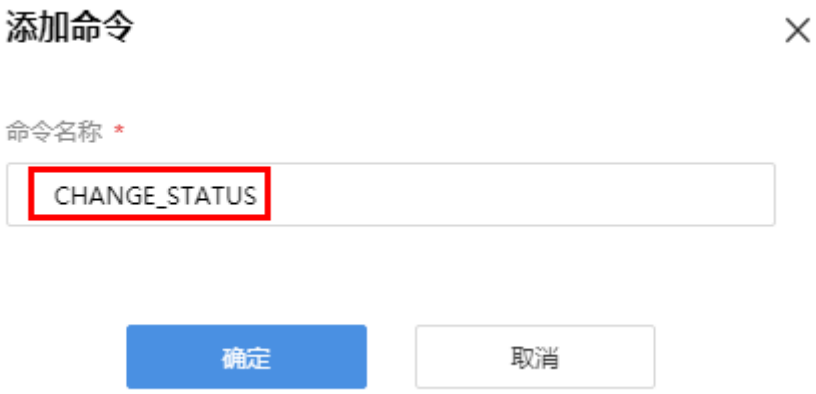
2. 点击“新增属性”，在弹出窗口中配置属性的各项参数，点击“确定”。“属性名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如batteryLevel、internalTemperature；其余参数，请根据此类设备的实际情况进行配置。

“数据类型”的配置可参考如下原则：

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



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

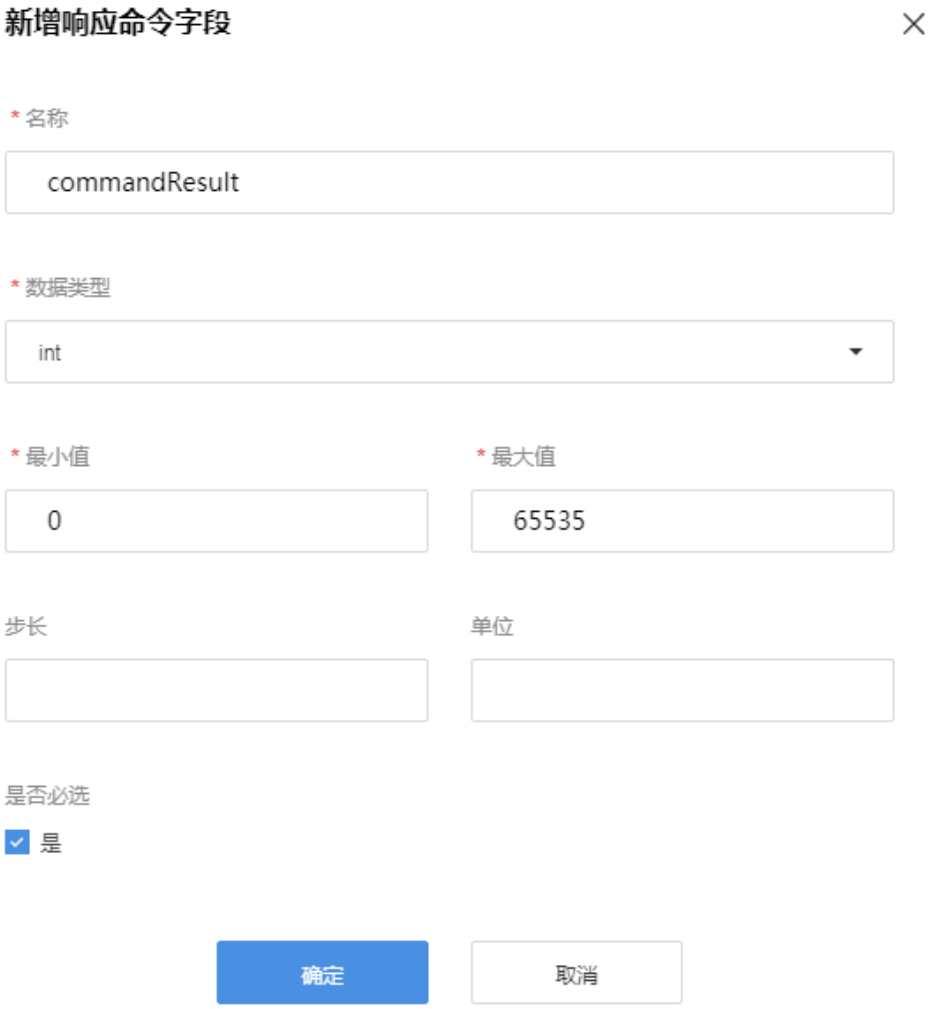


4. 点击“添加下发字段”，在弹出窗口中配置下发命令字段的各项参数，点击“确定”。“命令字段名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如statusValue；其余参数，请根据此类设备的实际情况进行配置。

5. 点击“添加响应字段”，在弹出窗口中配置响应命令字段的各项参数，点击“确定”。“响应字段名称”采用第一个单词首字母小写，其余单词的首字母大写的

命名方式，比如commandResult；其余参数，请根据此类设备的实际情况进行配置。

响应字段非必配参数，当需要设备返回命令执行结果时，才需要定义此字段。



----结束

1.3.2 线下开发指导

1.3.2.1 设备 Profile 写作

设备的Profile文件为json格式的文件，需要包含如下信息：

设备型号识别属性：设备类型、厂商、型号、协议类型。

服务列表：具体的功能服务说明列表。

命名规范

在Profile文件的开发过程中，需要遵循如下命名规范：

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

说明

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

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

设备 Profile

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

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

说明

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

设备型号识别属性：

属性	Profile中key	属性值
设备类型	deviceType	WaterMeter
制造商ID	manufacturerId	TestUtf8ManuId
制造商名称	manufacturerName	HZYB
设备型号	model	NBIoTDevic
协议类型	protocolType	CoAP

设备的服务列表

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

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

说明

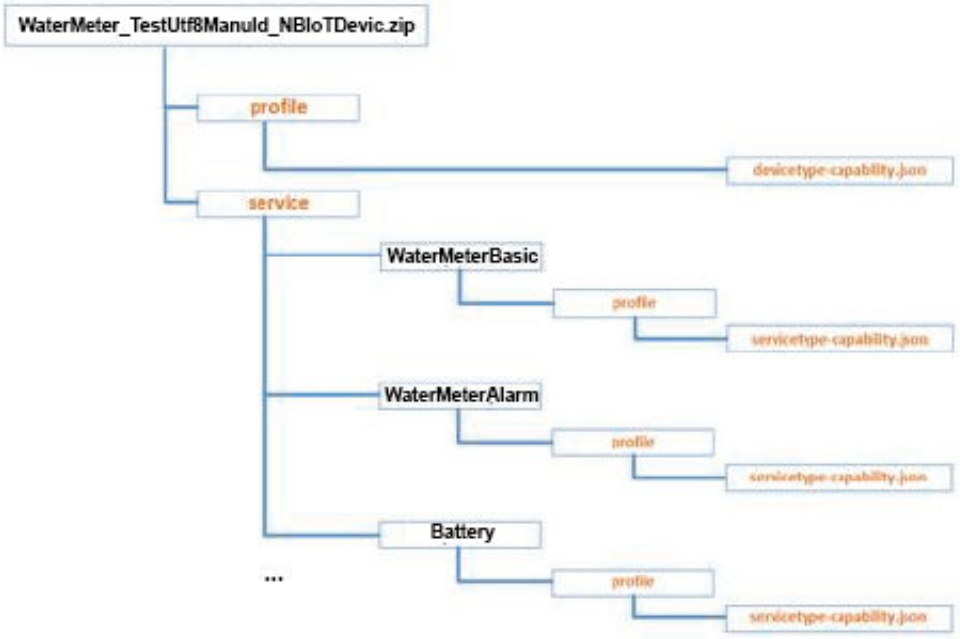
开发者可以通过咨询物联网平台支撑人员，以判断物联网平台是否支持自己的设备类型。如果开发者的设备类型或服务类型已经支持，则开发者可以向物联网平台支撑人员获取设备及服务的Profile文件参考。

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

Profile 打包

Profile写作完成后，需要按如下层级结构打包：

图 1-1 Profile 层级结构



Profile打包需要遵循如下几点要求：

- Profile文件的目录层级结构必须如图1-1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1-1中橙色的命名不能改动。
- Profile文件以zip形式压缩。
- Profile文件的命名必须按照deviceType_manufacturerId_model的格式命名，其中的deviceType、manufacturerId、model必须与devicetype-capability.json中对应字段的定义一致。例如：本实例中devicetype-capability.json的主要字段如下：

```
{
    "devices": [
        {
            "manufacturerId": "TestUtf8ManuId",
            "manufacturerName": "HZYB",
            "model": "NBIoTDevice",
            "protocolType": "CoAP",
            "deviceType": "WaterMeter",
            "serviceTypeCapabilities": ****
        }
    ]
}
```

- 图1-1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。

1.3.2.2 设备 Profile 提供形式

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

1.3.2.3 设备 Profile 文件字段含义说明

设备能力

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

```
{
    "devices": [
        {
            "manufacturerId": "TestUtf8ManuId",
            "manufacturerName": "HZYB",
            "model": "NBIoTDevice",
            "protocolType": "CoAP",
            "deviceType": "WaterMeter",
            "omCapability":{
                        "upgradeCapability" : {
                              "supportUpgrade":true,
                              "upgradeProtocolType":"PCP"
                        },
                        "fwUpgradeCapability" : {
                              "supportUpgrade":true,
                              "upgradeProtocolType":"LWM2M"
                        },
                        "configCapability" : {
                              "supportConfig":true,
                              "configMethod":"file",
                              "defaultConfigFile": {
                                 "waterMeterInfo" : {
                                      "waterMeterPirTime" : "300"
                                  }
```

```
                            }
                        }
                },
                "serviceTypeCapabilities": [
                    {
                        "serviceId": "WaterMeterBasic",
                        "serviceType": "WaterMeterBasic",
                        "option": "Mandatory"
                    },
                    {
                        "serviceId": "WaterMeterAlarm",
                        "serviceType": "WaterMeterAlarm",
                        "option": "Mandatory"
                    },
                    {
                        "serviceId": "Battery",
                        "serviceType": "Battery",
                        "option": "Optional"
                    },
                    {
                        "serviceId": "DeliverySchedule",
                        "serviceType": "DeliverySchedule",
                        "option": "Mandatory"
                    },
                    {
                        "serviceId": "Connectivity",
                        "serviceType": "Connectivity",
                        "option": "Mandatory"
                    }
                ]
            }
    ]
}
```

各字段的解释：

字段	子字段		可选/必选	描述
devices			必选	包含了一个设备的完整能力信息（根节点不能修改）。
	manufacturerId		必选	指示设备的制造商ID。
	manufacturerName		必选	指示设备的制造商名称 （只允许英文）。
	model		必选	指示设备的型号，考虑到一款设备下的多种型号，建议包含字母或数字以保证可扩展性。
	protocolType		必选	指示设备接入物联网平台的协议类型。如NB-IoT的设备取值为CoAP。
	deviceType		必选	指示设备的类型。
	omCapability		可选	定义设备的软件升级、固件升级和配置更新的能力，字段含义详情见下文中的：omCapability结构描述。 如果设备不涉及软件/固件升级，本字段可以删除。

字段	子字段		可选/必选	描述
	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。
	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。

字段	子字段	可选/必选	描述
	defaultConfigFile	可选	设备默认配置信息（Json格式），具体配置信息由设备商自定义。物联网平台只储存该信息供下发时使用，不解析处理配置字段的具体含义。

服务能力

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

```
{
    "services": [
        {
            "serviceType": "WaterMeterBasic",
            "description": "WaterMeterBasic",
            "commands": [
                {
                    "commandName": "SET_PRESSURE_READ_PERIOD",
                    "paras": [
                        {
                            "paraName": "value",
                            "dataType": "int",
                            "required": true,
                            "min": 1,
                            "max": 24,
                            "step": 1,
                            "maxLength": 10,
                            "unit": "hour",
                            "enumList": null
                        }
                    ],
                    "responses": [
                        {
                            "responseName": "SET_PRESSURE_READ_PERIOD_RSP",
                            "paras": [
                                {
                                    "paraName": "result",
                                    "dataType": "int",
                                    "required": true,
                                    "min": -1000000,
                                    "max": 1000000,
                                    "step": 1,
                                    "maxLength": 10,
                                    "unit": null,
                                    "enumList": null
                                }
                            ]
                        }
                    ]
                }
            ],
            "properties": [
                {
                    "propertyName": "registerFlow",
                    "dataType": "int",
                    "required": true,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "R",
                    "unit": null,
```

```
            "enumList": null
        },
        {
            "propertyName": "currentReading",
            "dataType": "string",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "timeOfReading",
            "dataType": "string",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": null,
            "enumList": null
        },
        {
            "propertyName": "internalTemperature",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "0.01° C",
            "enumList": null
        },
        {
            "propertyName": "dailyFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "dailyReverseFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "peakFlowRate",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
```



```
                    "maxLength": 0,
                    "method": "W",
                    "unit": "L/H",
                    "enumList": null
                },
                {
                    "propertyName": "peakFlowRateTime",
                    "dataType": "string",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "W",
                    "unit": null,
                    "enumList": null
                },
                {
                    "propertyName": "intervalFlow",
                    "dataType": "array",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "W",
                    "unit": "L",
                    "enumList": null
                },
                {
                    "propertyName": "pressure",
                    "dataType": "array",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "W",
                    "unit": "kPa",
                    "enumList": null
                },
                {
                    "propertyName": "temperature",
                    "dataType": "array",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "W",
                    "unit": "0.01° C",
                    "enumList": null
                },
                {
                    "propertyName": "vibration",
                    "dataType": "array",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "W",
                    "unit": "0.01g",
                    "enumList": null
                }
            ]
        }
    ]
}
```


各字段的解释：

字段	子字段				必选/可选	描述
services					必选	包含了一个服务的完整信息（根节点不可修改）。
	serviceType				必选	指示服务的类型，与devicetype-capability.json中serviceType字段保持一致。
	description				必选	指示服务的描述信息。 非功能性字段，仅起到描述作用，可置为null。
	commands				必选	指示设备可以执行的命令，如果本服务无命令则置null。
		commandName			必选	指示命令的名字，命令名与参数共同构成一个完整的命令。
		paras			必选	命令包含的参数。
			paraName		必选	命令中参数的名字。
			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 上报数据时，复杂类型数据格式如下： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传
			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。

字段	子字段				必选/可选	描述
			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。
		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。
				paraName	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： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’ 如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传

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
		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： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’ 如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传
		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。

1.3.3 参考信息

1.3.3.1 产品模型样例

附录一 WaterMeter Profile 样例

样例由六个文件构成，文件名和文件内容如下。

1. **devicetype-capability.json**

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


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

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

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

```
{
    "services": [
        {
            "serviceType": "Connectivity",
            "description": "Connectivity",
            "commands": null,
            "properties": [
                {
                    "propertyName": "signalStrength",
                    "dataType": "int",
                    "required": true,
```



```
                    "min": -110,
                    "max": -48,
                    "step": 1,
                    "maxLength": 0,
                    "method": "RE",
                    "unit": "dbm",
                    "enumList": null
                },
                {
                    "propertyName": "linkQuality",
                    "dataType": "int",
                    "required": false,
                    "min": -110,
                    "max": -48,
                    "step": 1,
                    "maxLength": 0,
                    "method": "RE",
                    "unit": "dbm",
                    "enumList": null
                },
                {
                    "propertyName": "cellId",
                    "dataType": "int",
                    "required": false,
                    "min": 0,
                    "max": 268435455,
                    "step": 1,
                    "maxLength": 0,
                    "method": "RE",
                    "unit": null,
                    "enumList": null
                }
            ]
        }
    ]
}
```

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

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

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

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

```
{
    "services": [
        {
            "serviceType": "WaterMeterAlarm",
            "description": "WaterMeterAlarm",
            "commands": null,
            "properties": [
                {
                    "propertyName": "lowFlowAlarm",
                    "dataType": "int",
                    "required": true,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "RE",
                    "unit": null,
```

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

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



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



```
        }
    ]
}
```

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

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

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



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

1.3.3.2 产品模型各字段含义

设备能力

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

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

```
                    "serviceType": "Battery",
                    "option": "Optional"
                },
                {
                    "serviceId": "DeliverySchedule",
                    "serviceType": "DeliverySchedule",
                    "option": "Mandatory"
                },
                {
                    "serviceId": "Connectivity",
                    "serviceType": "Connectivity",
                    "option": "Mandatory"
                }
            ]
        }
    ]
}
```

各字段的解释：

字段	子字段		可选/必选	描述
devices			必选	包含了一个设备的完整能力信息（根节点不能修改）。
	manufacturerId		必选	指示设备的制造商ID。
	manufacturerName		必选	指示设备的制造商名称 （只允许英文）。
	model		必选	指示设备的型号，考虑到一款设备下的多种型号，建议包含字母或数字以保证可扩展性。
	protocolType		必选	指示设备接入物联网平台的协议类型。如NB-IoT的设备取值为CoAP。
	deviceType		必选	指示设备的类型。
	omCapability		可选	定义设备的软件升级、固件升级和配置更新的能力，字段含义详情见下文中的：omCapability结构描述。 如果设备不涉及软件/固件升级，本字段可以删除。
	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。

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

服务能力

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

```
{
   "services": [
```



```
    {
        "serviceType": "WaterMeterBasic",
        "description": "WaterMeterBasic",
        "commands": [
            {
                "commandName": "SET_PRESSURE_READ_PERIOD",
                "paras": [
                    {
                        "paraName": "value",
                        "dataType": "int",
                        "required": true,
                        "min": 1,
                        "max": 24,
                        "step": 1,
                        "maxLength": 10,
                        "unit": "hour",
                        "enumList": null
                    }
                ],
                "responses": [
                    {
                        "responseName": "SET_PRESSURE_READ_PERIOD_RSP",
                        "paras": [
                            {
                                "paraName": "result",
                                "dataType": "int",
                                "required": true,
                                "min": -1000000,
                                "max": 1000000,
                                "step": 1,
                                "maxLength": 10,
                                "unit": null,
                                "enumList": null
                            }
                        ]
                    }
                ]
            }
        ],
        "properties": [
            {
                "propertyName": "registerFlow",
                "dataType": "int",
                "required": true,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "R",
                "unit": null,
                "enumList": null
            },
            {
                "propertyName": "currentReading",
                "dataType": "string",
                "required": false,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "W",
                "unit": "L",
                "enumList": null
            },
            {
                "propertyName": "timeOfReading",
                "dataType": "string",
                "required": false,
                "min": 0,
```



```
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": null,
            "enumList": null
        },
        {
            "propertyName": "internalTemperature",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "0.01° C",
            "enumList": null
        },
        {
            "propertyName": "dailyFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "dailyReverseFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "peakFlowRate",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": "L/H",
            "enumList": null
        },
        {
            "propertyName": "peakFlowRateTime",
            "dataType": "string",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "W",
            "unit": null,
            "enumList": null
        },
        {
            "propertyName": "intervalFlow",
```



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

各字段的解释：

字段	子字段				必选/可选	描述
services					必选	包含了一个服务的完整信息（根节点不可修改）。
	serviceType				必选	指示服务的类型，与devicetype-capability.json中serviceType字段保持一致。

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。
		paras			必选	命令包含的参数。
			paraName		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 上报数据时，复杂类型数据格式如下： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传
			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。
			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
			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。
				paraName	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： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传
				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。

字段	子字段				必选/可选	描述
				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。
		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： ● string list：["str1","str2","str3"] ● DateTime：yyyyMMdd’T’HHmmss’Z’如:20151212T121212Z ● jsonObject：自定义json结构体，物联网平台不解析，仅进行透传

字段	子字段				必选/可选	描述
		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。
		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。

1.4 开发编解码插件

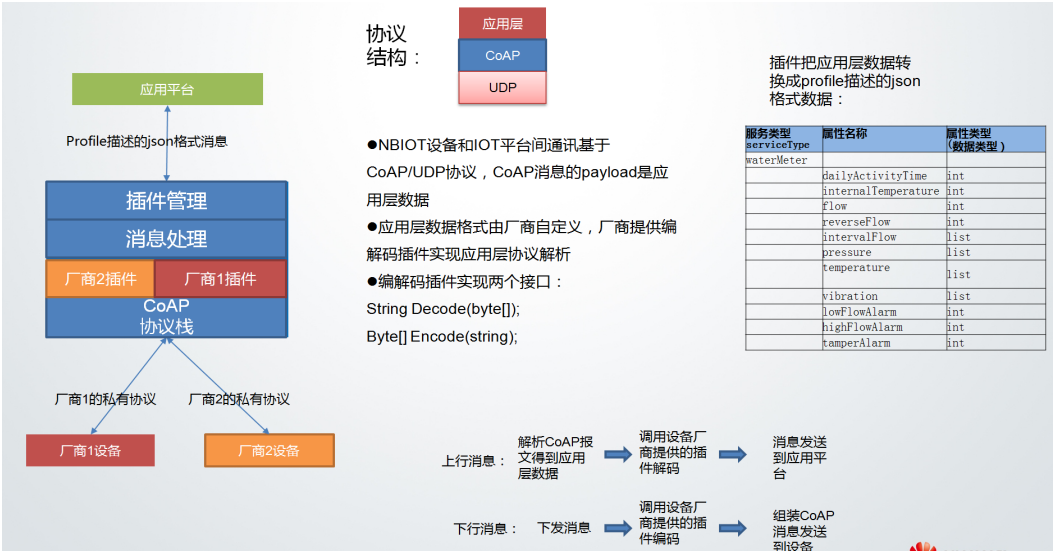
1.4.1 开发指导

概述

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

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

图 1-2 整体方案



操作指导

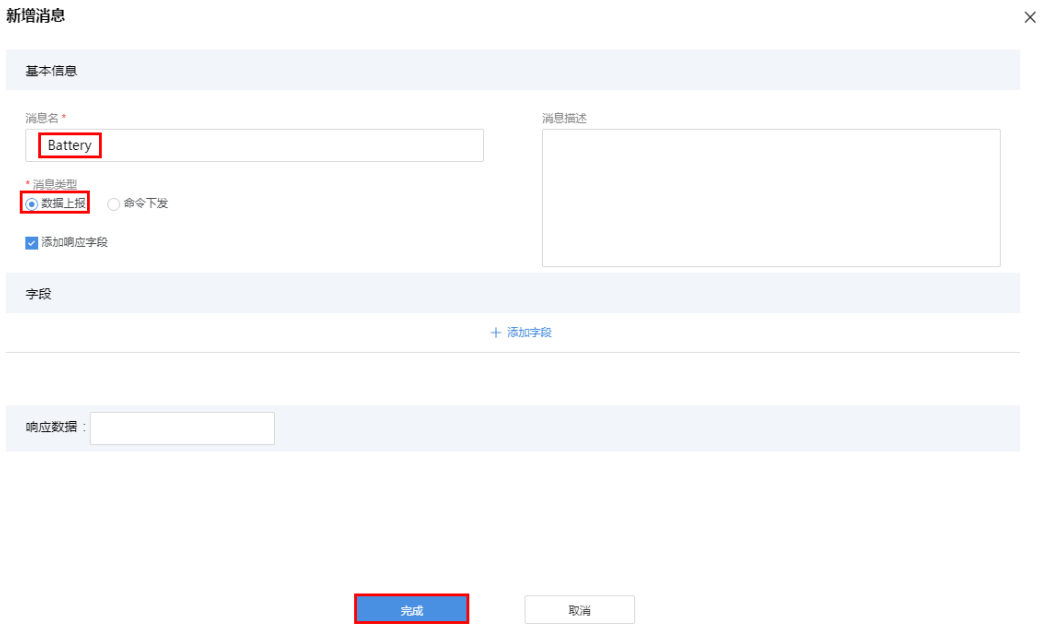
在创建项目和产品时：如果选择使用系统模板，部分模板会包含编解码插件，开发者可以直接使用或在此基础上进行修改；如果选择自定义产品模板，则需要完整开发编解码插件，操作如下：

步骤1 在产品开发空间，点击“编解码插件开发”。

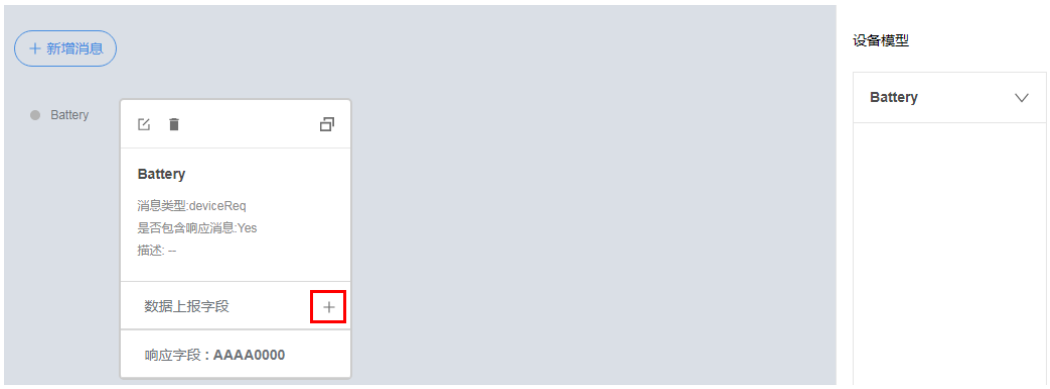
步骤2 在“在线编解码插件编辑器”区域，点击“新增消息”。

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

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



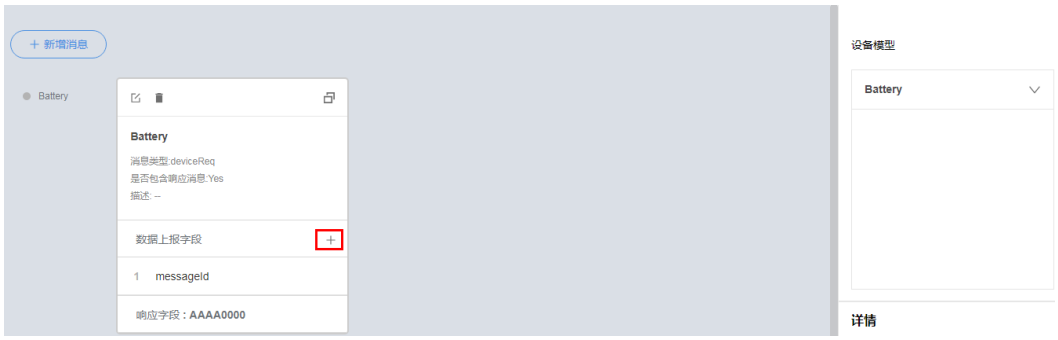
步骤4 点击数据上报字段后的“+”。



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

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

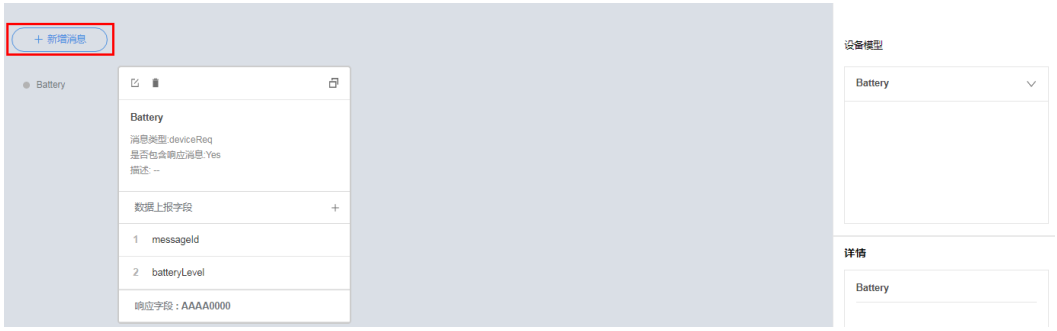
步骤6 点击数据上报字段后的“+”。



步骤7 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

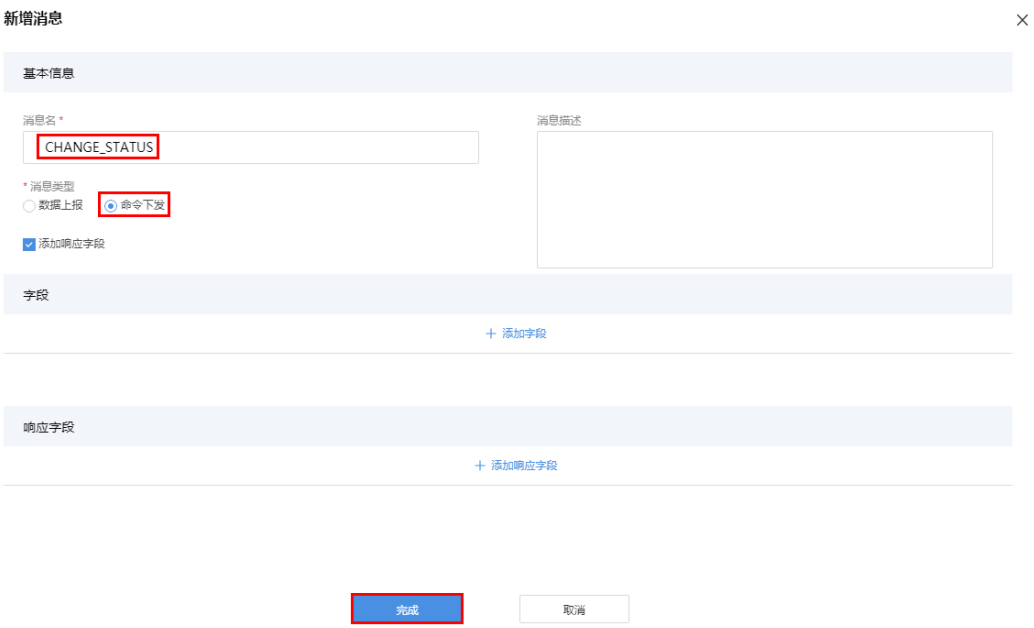
- “字段名”只能输入包含字母、数字、_和$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

步骤8 在“编解码插件编辑器”区域，点击“新增消息”。

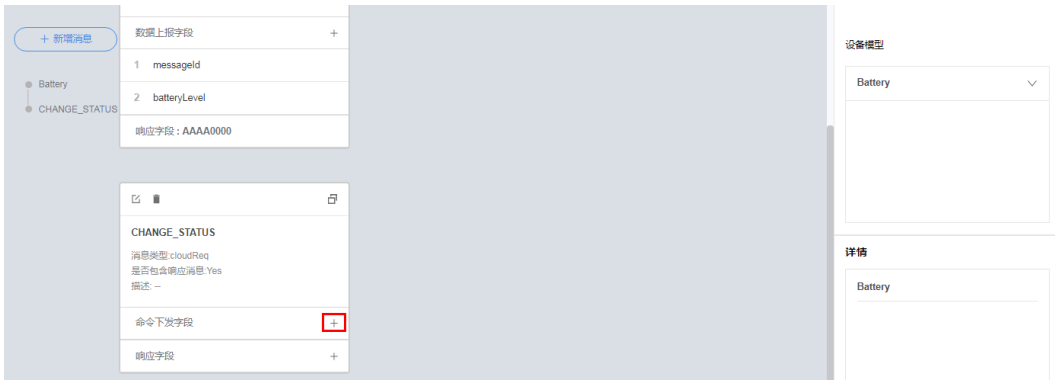


步骤9 系统将弹出“新建消息”窗口，填写“消息名”，“消息类型”选择“命令下发”，点击“完成”。

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



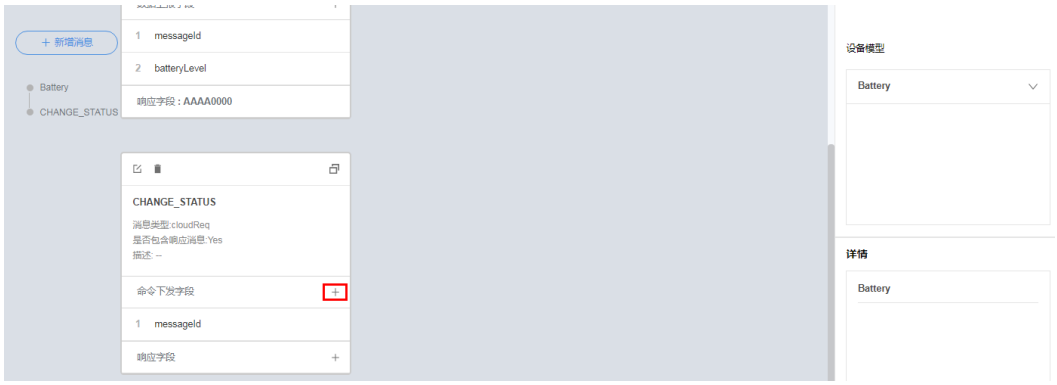
步骤10 点击命令下发字段后的“+”。



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

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

步骤12 点击命令下发字段后的“+”。



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

步骤14 点击命令下发后的“+”。

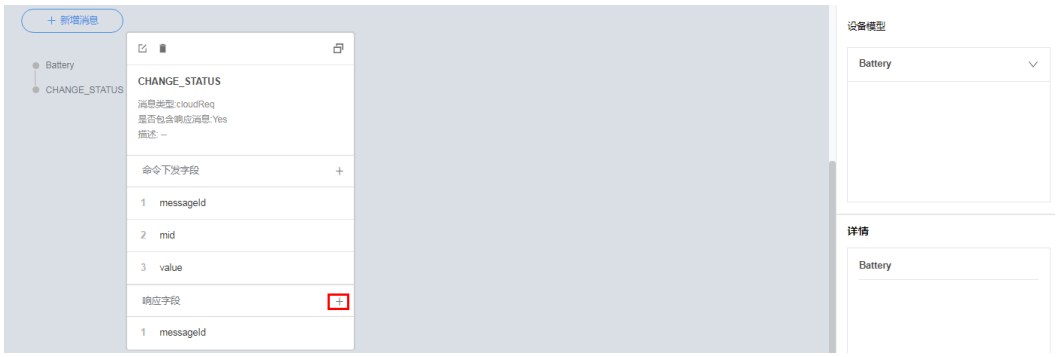
步骤15 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

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

步骤16 点击响应字段后的“+”。

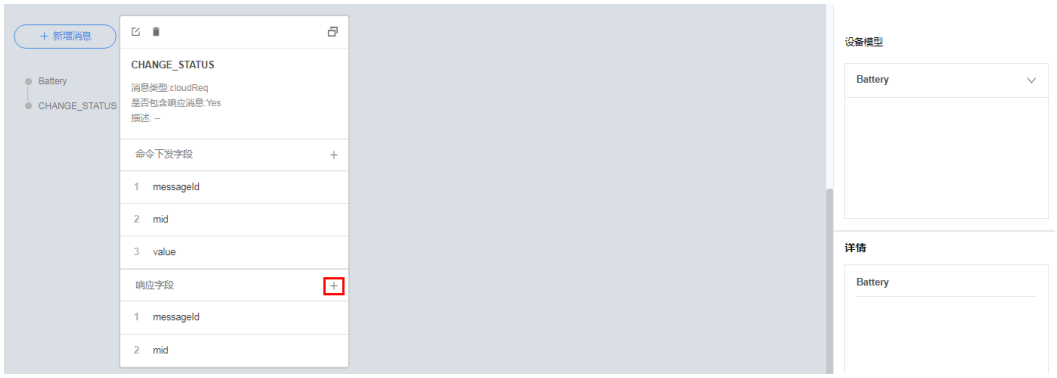
步骤17 系统将弹出“增加字段”窗口，勾选“标记为地址域”，其余参数将自动填充，点击“完成”。

步骤18 点击响应字段后的“+”。



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

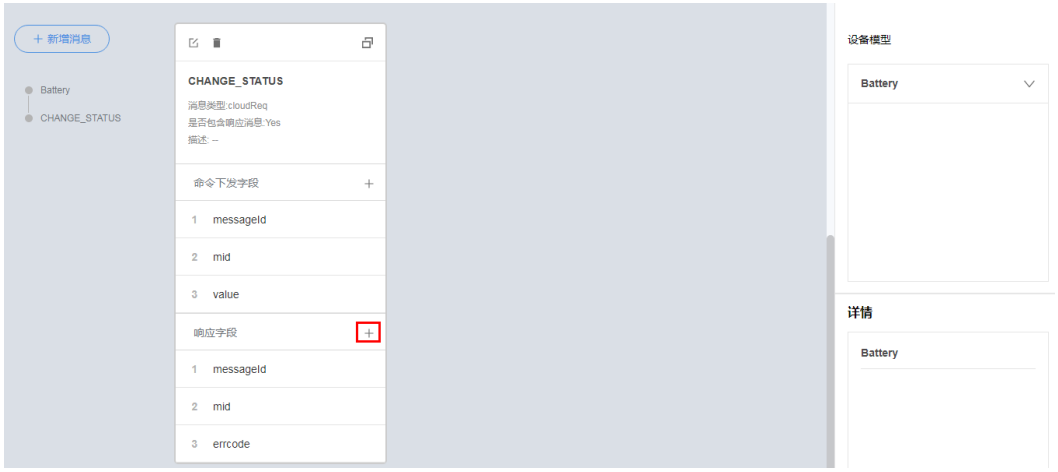
步骤20 点击响应字段后的“+”。



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

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

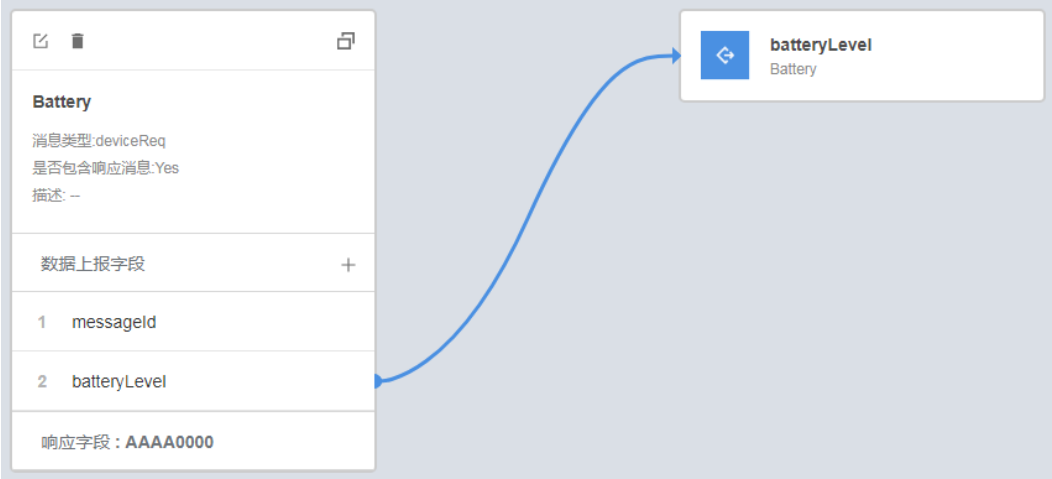
步骤22 点击响应字段后的“+”。

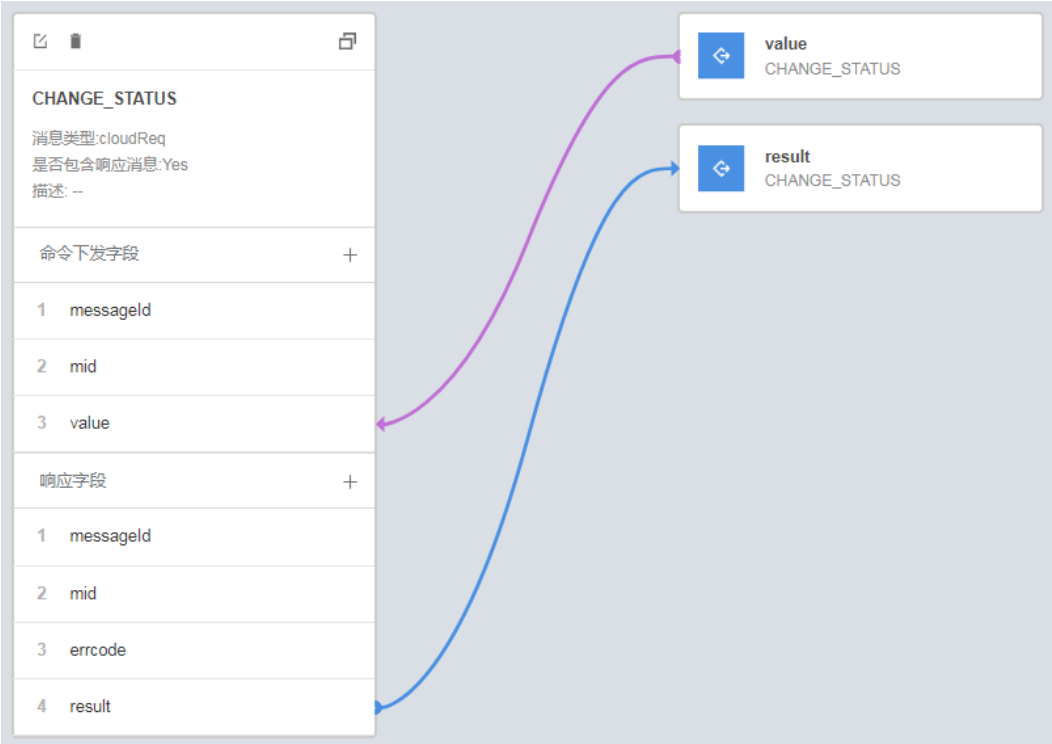


步骤23 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

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

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





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

----结束

1.4.2 线下开发指导

1.4.2.1 开发环境准备

下载 eclipse

下载Eclipse安装包，直接解压缩到本地即可使用。

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

下载 maven 插件

下载Maven插件包（zip格式），直接解压缩到本地。

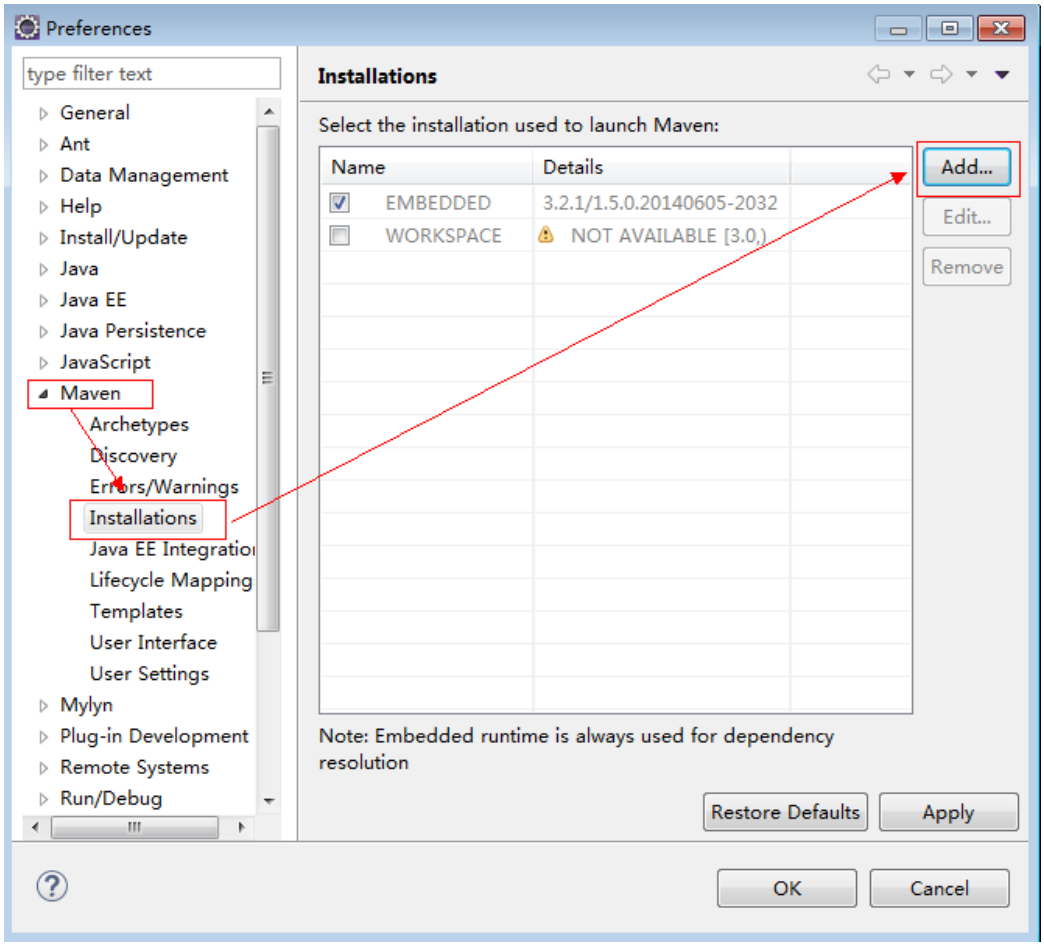
官网下载地址：**http://maven.apache.org/download.cgi**

配置 Maven 插件

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

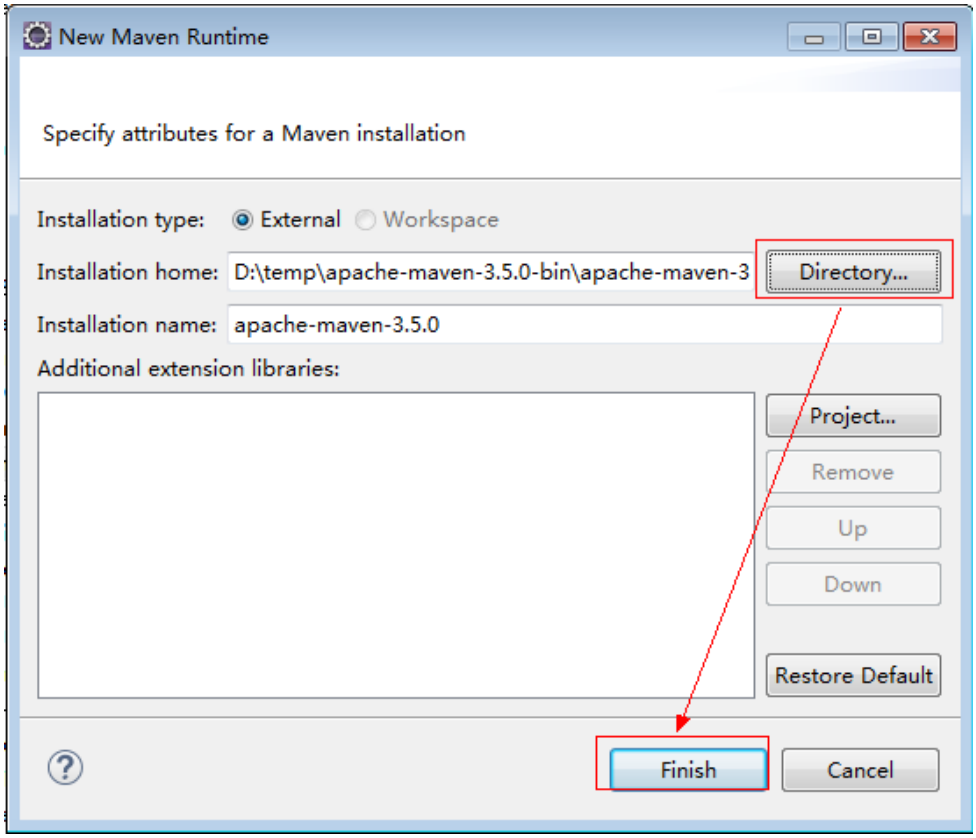
步骤1 选择Eclipse菜单“Windows”->“Preferences”，打开Preferences窗口，选择“Maven”->“Installations”->“Add”。

图 1-3 配置 Maven 插件 1



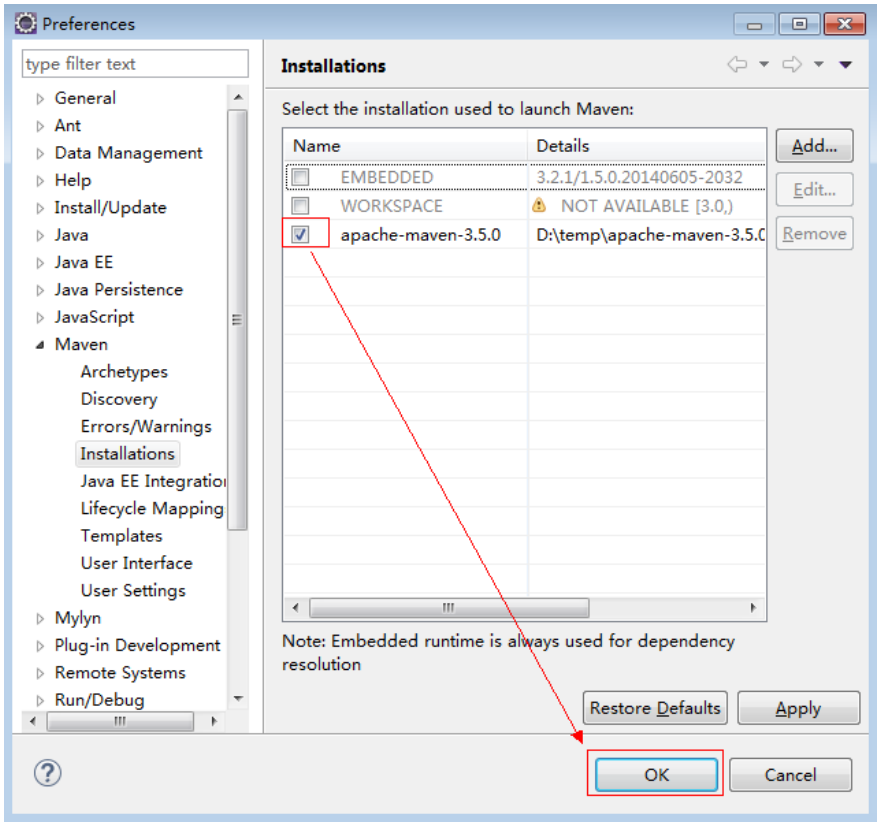
步骤2 选择maven插件包路径，点击“Finish”，导入Maven插件。

图 1-4 配置 Maven 插件 2



步骤3 选择导入的maven插件，点击“OK”。

图 1-5 配置 Maven 插件 3



说明

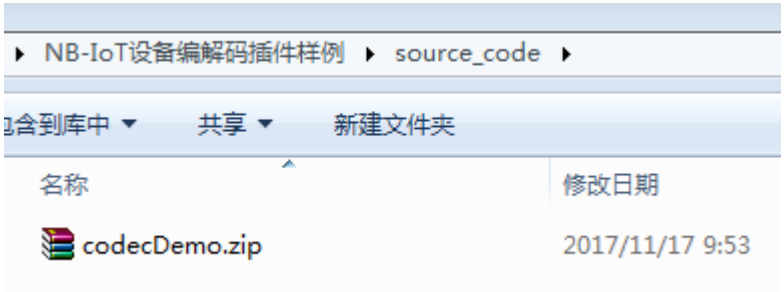
JDK的安装和Java环境变量的配置请参见**安装JDK1.8和配置Java环境变量（Windows操作系统）**。

----结束

1.4.2.2 导入编解码插件 DEMO 工程

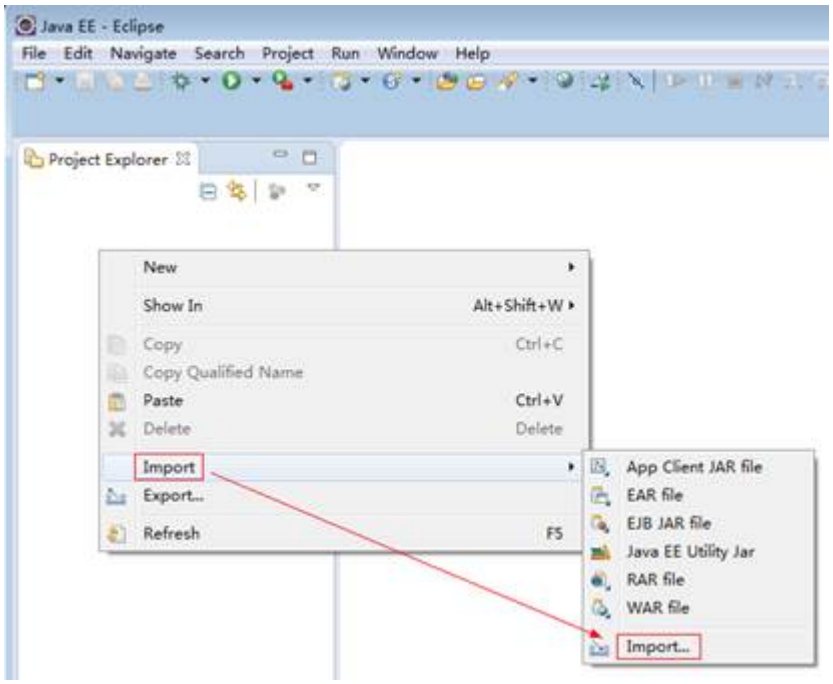
步骤1 下载**编解码插件DEMO工程**，在“source_code”文件夹中获取“codecDemo.zip”，将其解压到本地。

图 1-6 编解码插件 DEMO 的位置



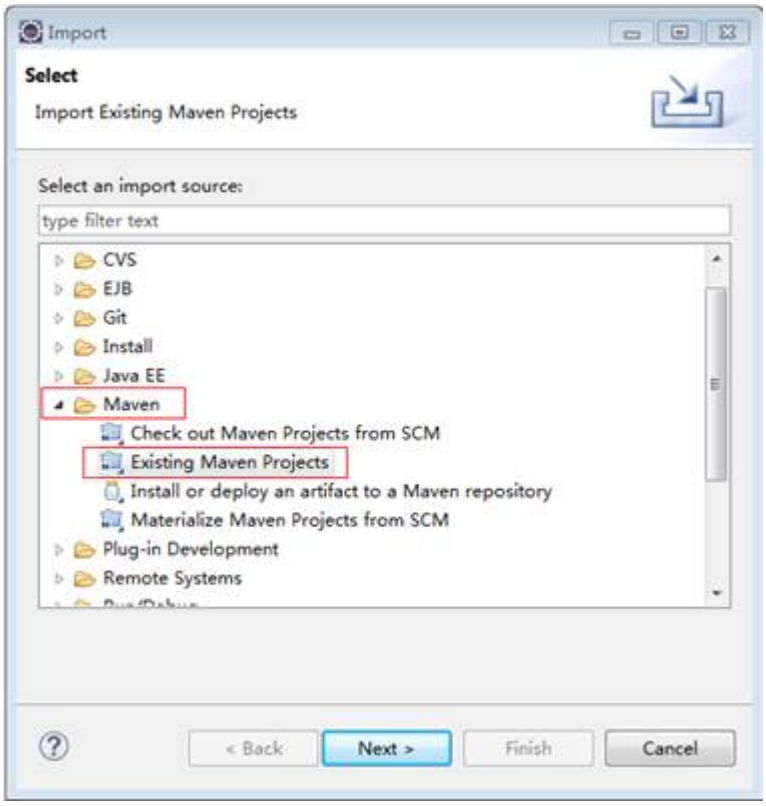
步骤2 打开Eclipse，右击Eclipse左侧“Project Explorer”空白处，选择“Import > Import...”。

图 1-7 导入 DEMO 工程 1



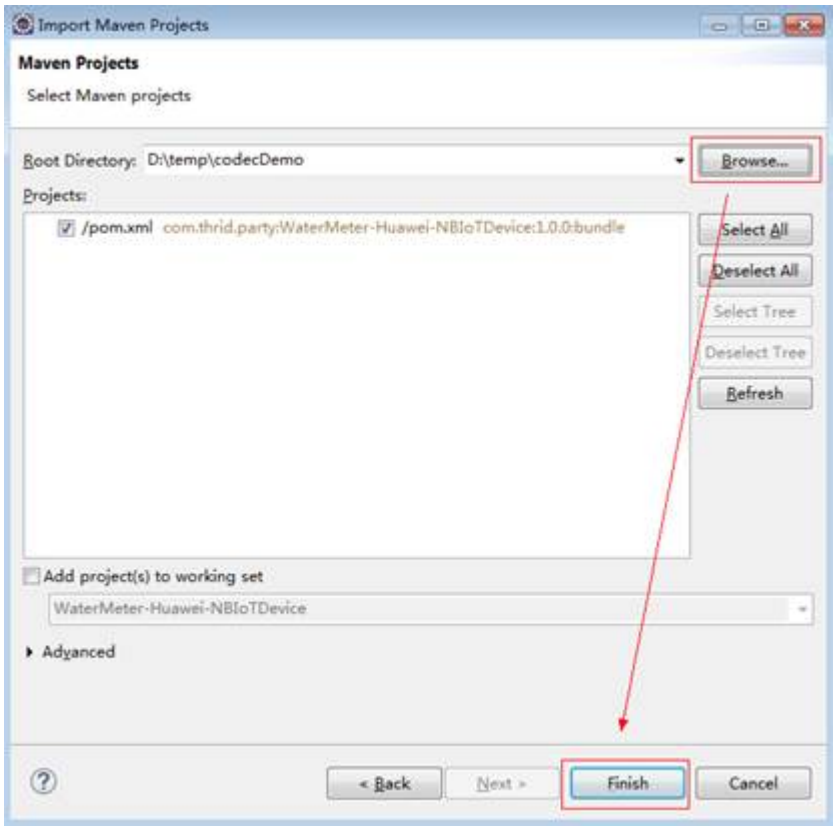
步骤3 展开“Maven”，选择“Existing Maven Projects”，点击“Next”。

图 1-8 导入 DEMO 工程 2



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

图 1-9 导入 DEMO 工程 3



----结束

1.4.2.3 开发插件

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

1.4.2.4 编解码插件打包

本节介绍编解码完成后如何打包和制作插件包。

编解码 Maven 打包

完成插件编程后使用Maven进行打包，Windows中步骤如下：

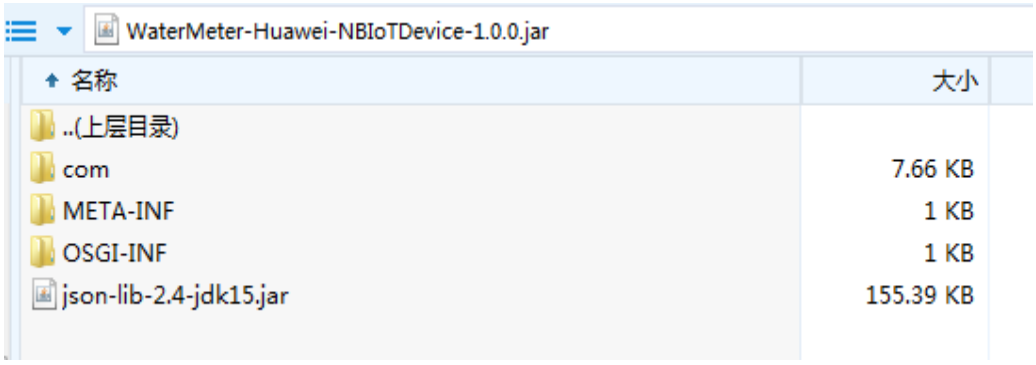
步骤1 打开DOS窗口，进入“pom.xml”所在的目录。

步骤2 输入maven打包命令：mvn package。

步骤3 DOS窗口中显示“BUILD SUCCESS”后，打开与“pom.xml”目录同级的target文件夹，获取打包好的jar包。

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

图 1-10 Jar 包结构图



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

----结束

制作插件包

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

步骤2 将打包好的jar包放到“preload/”文件夹。

图 1-11 插件包结构图

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

说明

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

表 1-2 “package-info.json”字段说明

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

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

表 1-3 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":"NBIoTDevice",
    "description":"codec",
    "platform":"linux",
    "packageType":"CIGPlugin",
    "date":"2017-02-06 12:16:59",
```

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

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

说明

“package.zip”中不能包含“package”这层目录。

----结束

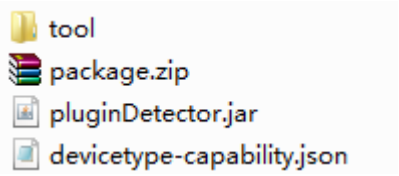
1.4.2.5 编解码插件质检

编解码插件的质检用于检验编解码是否可以正常使用。

步骤1 从物联网平台服务商获取**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。

图 1-12 文件准备目录



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

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

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

图 1-13 上报数据解码成功

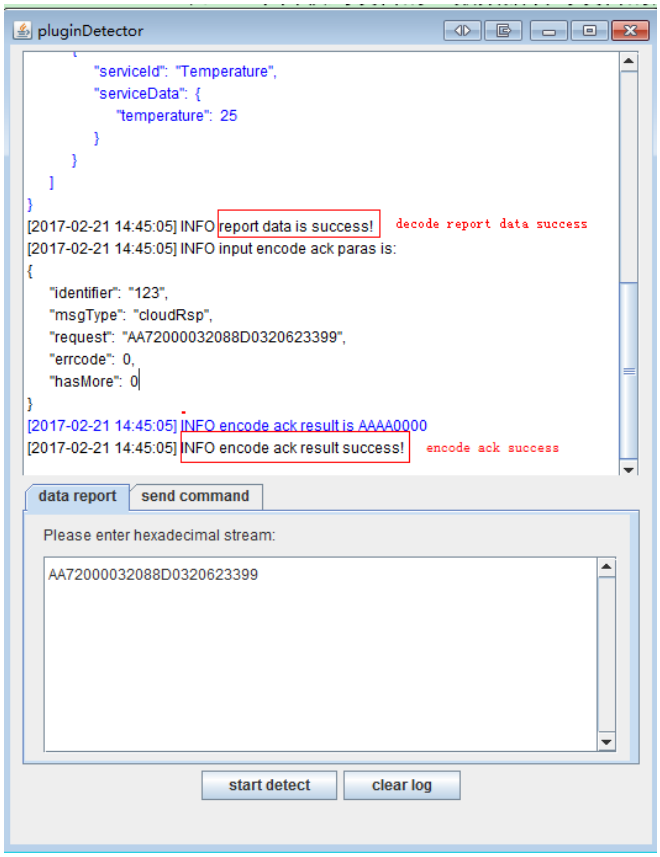
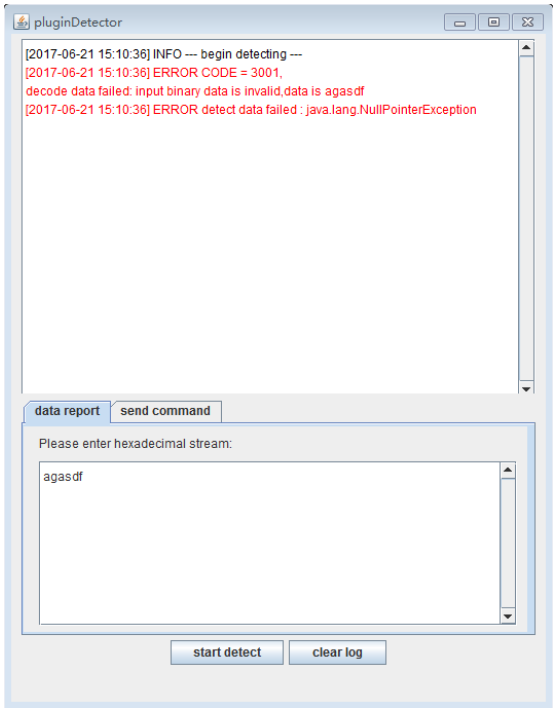


图 1-14 上报数据解码失败



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

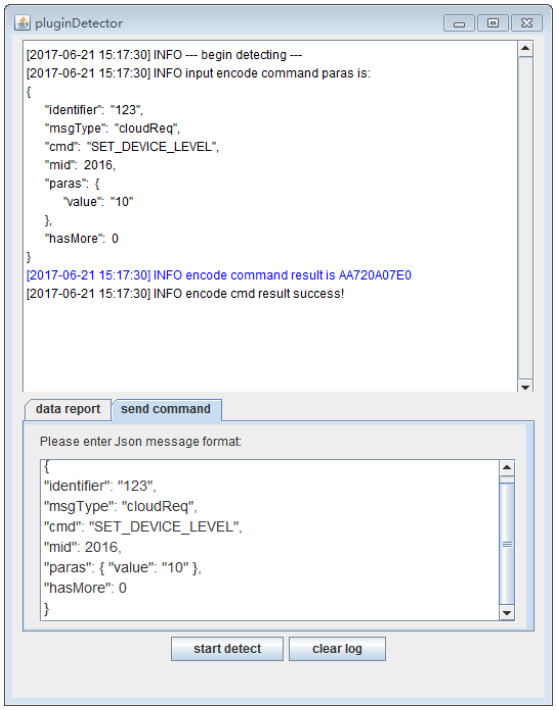
当提示“encode ack result success”时，表示对设备的应答消息编码成功。

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

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

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

图 1-15 编码控制命令下发成功



命令示例：

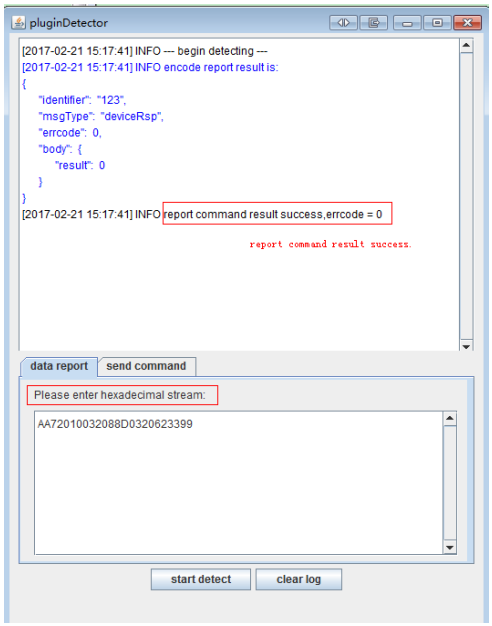
```
{
    "identifier": "123",
    "msgType": "cloudReq",
    "serviceId": "NBWaterMeterCommon",
    "cmd": "SET_DEVICE_LEVEL",
    "mid": 2016,
    "paras": {
        "value": "10"
    },
    "hasMore": 0
}
```

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

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

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

图 1-16 命令执行结果解码成功



----结束

1.4.2.6 编解码插件包离线签名

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

步骤1 登录物联网平台的管理门户。

步骤2 选择“管理 > 工具”，点击“离线签名工具”，获取离线签名工具。

图 1-17 下载离线签名工具

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

图 1-18 运行离线签名工具

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

请根据需要选择“签名算法”，当前提供两种签名算法：

- ECDSA_256K1+SHA256
- RSA2048+SHA256

设置“私钥加密口令”时，口令复杂度需要满足如下条件：

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

在保存目录下将生成公私密钥两个文件：

- 公钥文件：public.pem
- 私钥文件：private.pem

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

步骤6 选择需要数据签名的软件包，点击“进行数字签名”。

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

说明

离线签名工具只能对.zip格式的压缩包进行数字签名。

步骤7 在“软件包签名验证”区域导入公钥文件后，点击“确定”。

步骤8 选择需要签名验证的软件包（步骤6生成的带签名软件包），点击“进行软件包验签”。

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

说明

在进行软件包验签时，带签名软件包的存放路径不能包含中文字符。

----结束

1.4.3 插件开发示例

1.4.3.1 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

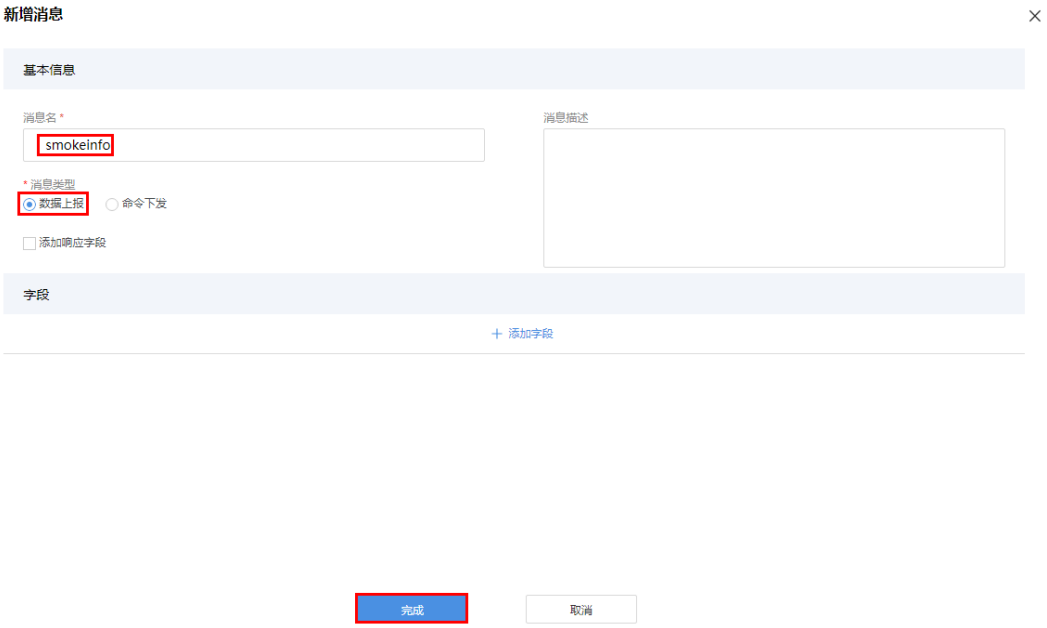
在烟感产品的开发空间，完成Profile定义。

- 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。

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。

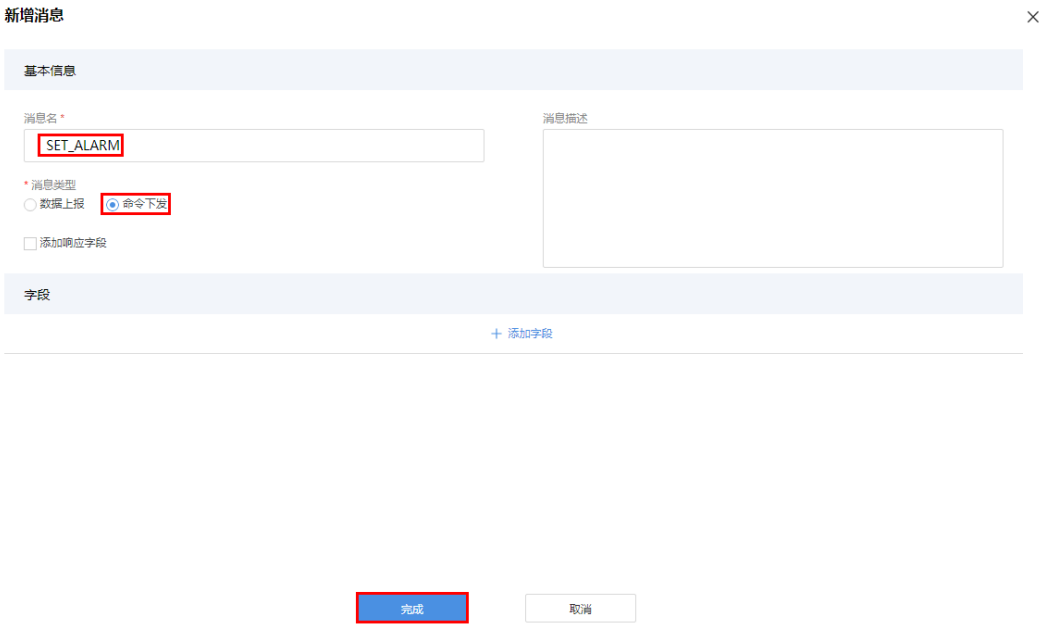


添加level字段，表示火灾级别。

- “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。

步骤3 配置命令下发消息。



添加value字段，表示下发命令的参数值。

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

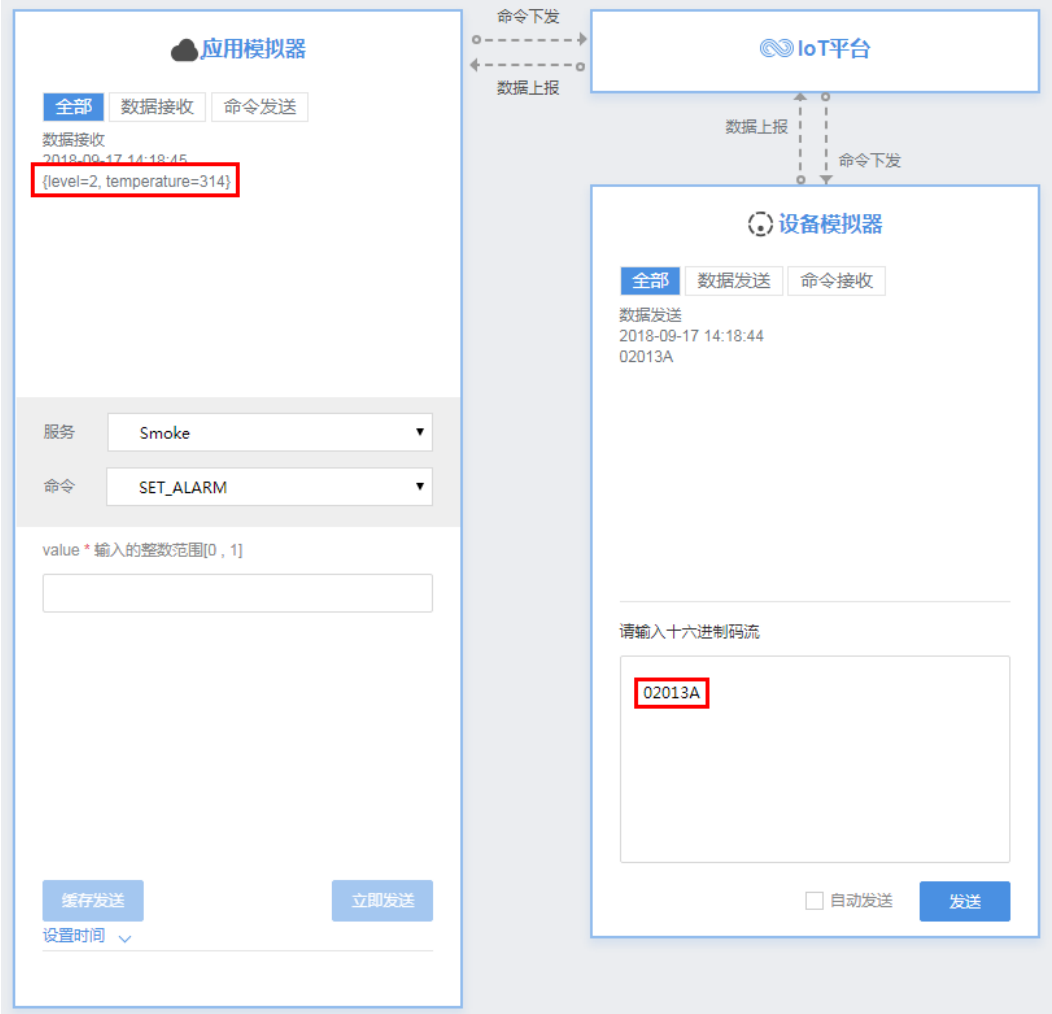
调测编解码插件

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：{ "serviceId": "Smoke", "method": "SET_ALARM", "paras": "{\"value\":1}" }。

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

----结束

1.4.3.2 多条数据上报消息

场景说明

有一款烟感设备，具有如下特征：

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

Profile 定义

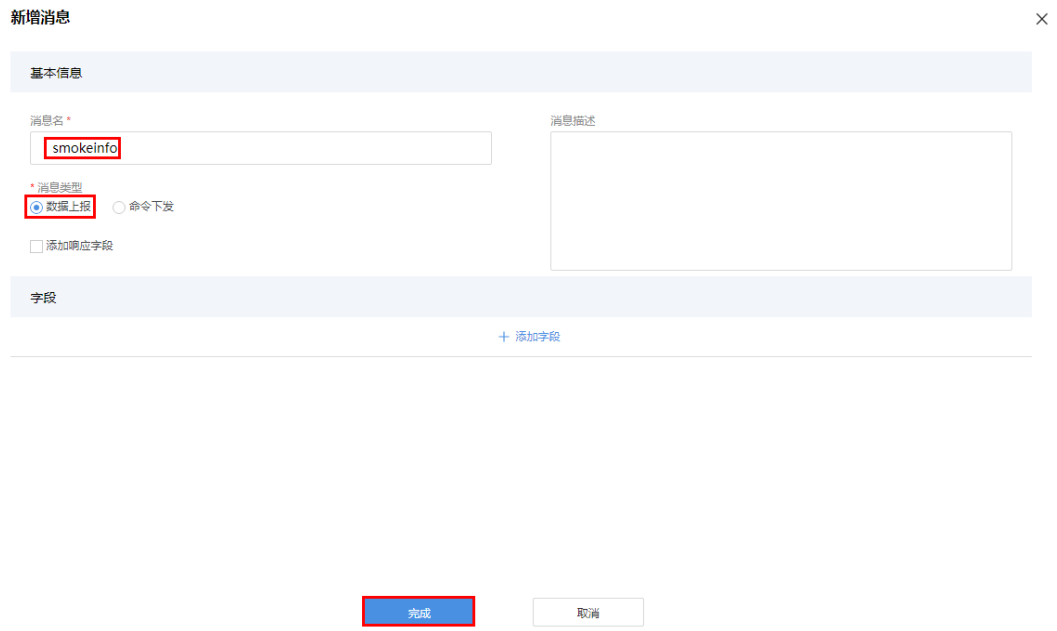
在烟感产品的开发空间，完成Profile定义。

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

编解码插件开发

步骤1 在烟感产品的开发空间，选择“编解码插件开发”。

步骤2 配置数据上报消息，上报火灾等级和温度。



添加messageId字段，表示消息种类。

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

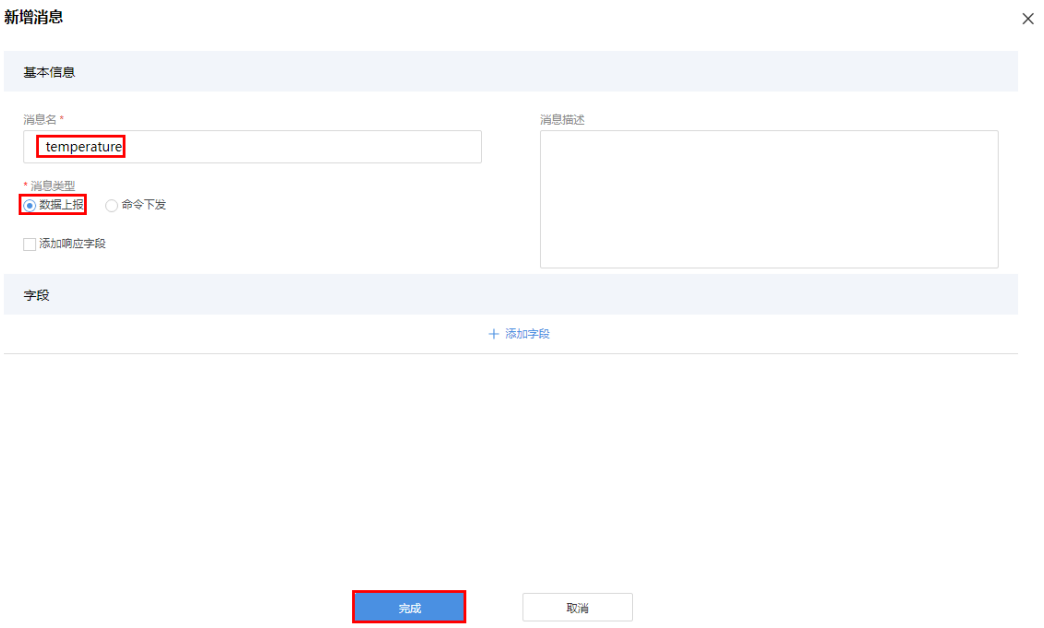
添加level字段，表示火灾级别。

- “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。

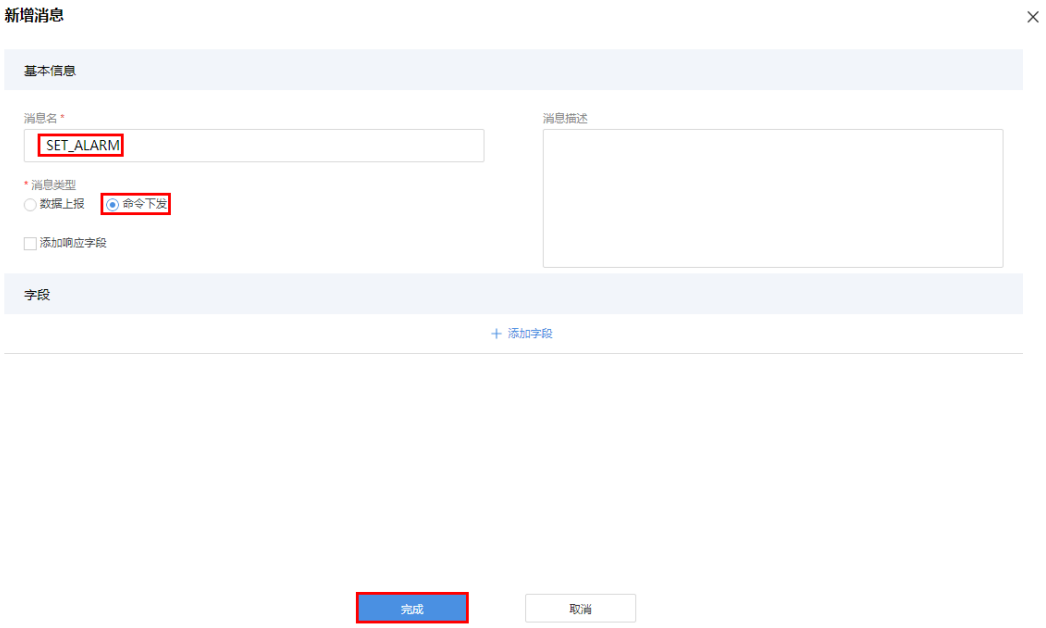
步骤3 配置数据上报消息，只上报温度。



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temperature字段，表示温度。

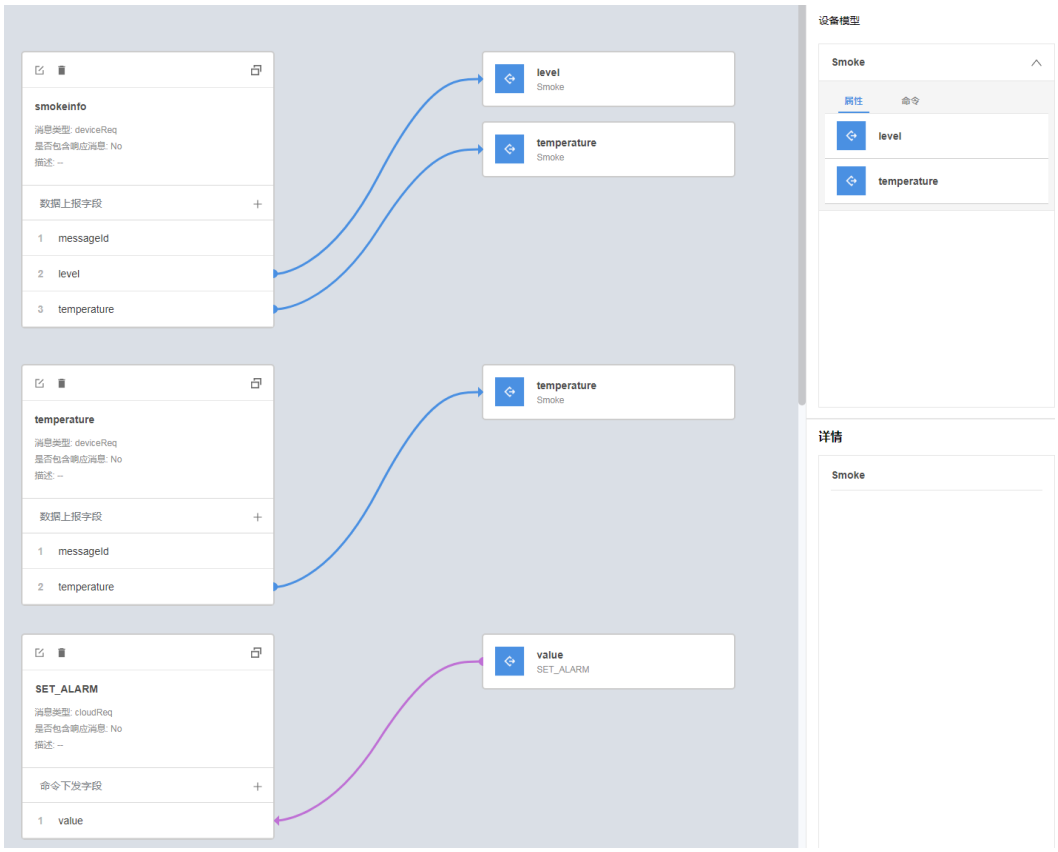
步骤4 配置命令下发消息。



添加value字段，表示下发命令的参数值。

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

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



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

----结束

调测编解码插件

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

勾选“没有真实的物理设备”，点击“创建”。

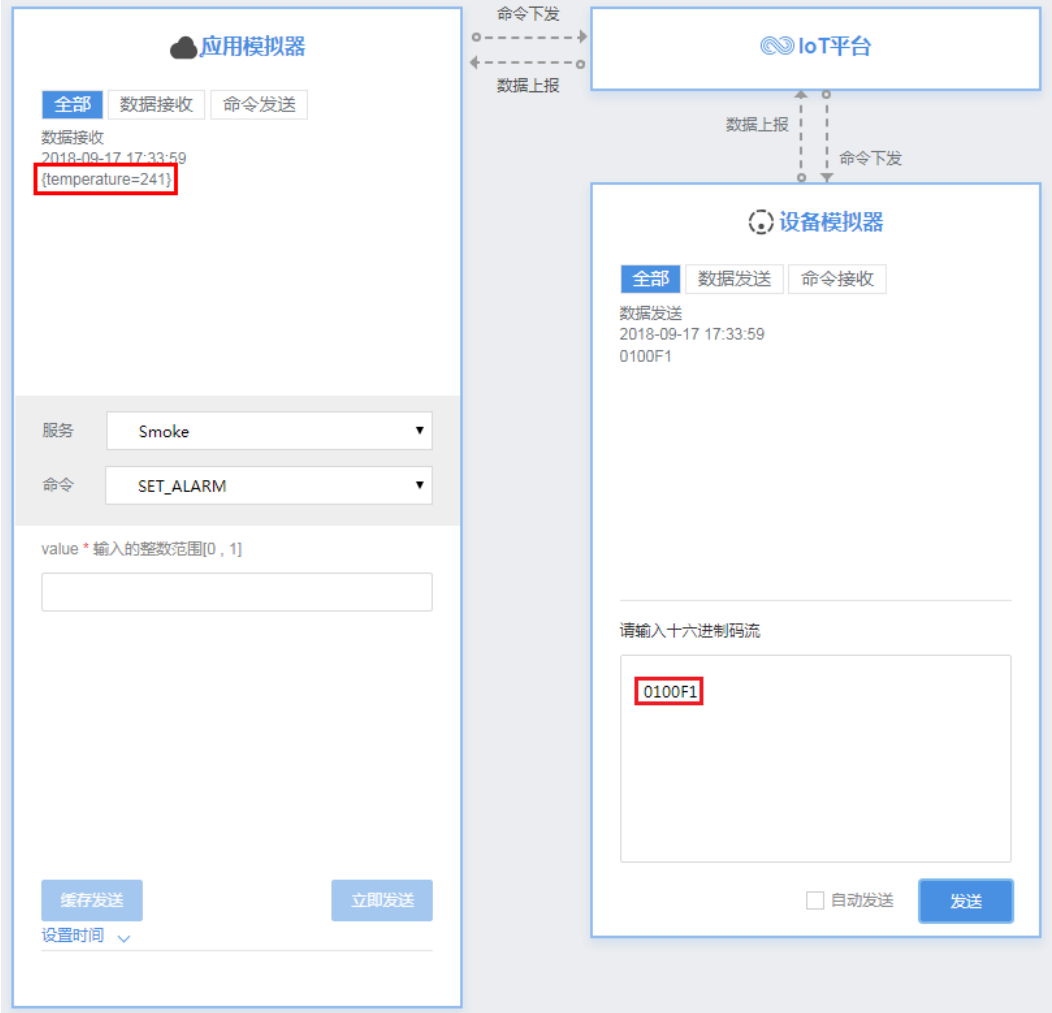
步骤2 使用设备模拟器进行数据上报。

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

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

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

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



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

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

----结束

1.4.3.3 字符串及可变长字符串数据类型

场景说明

有一款烟感设备，具有如下特征：

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

  说明

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

Profile 定义

在烟感产品的开发空间完成Profile定义。

编解码插件开发

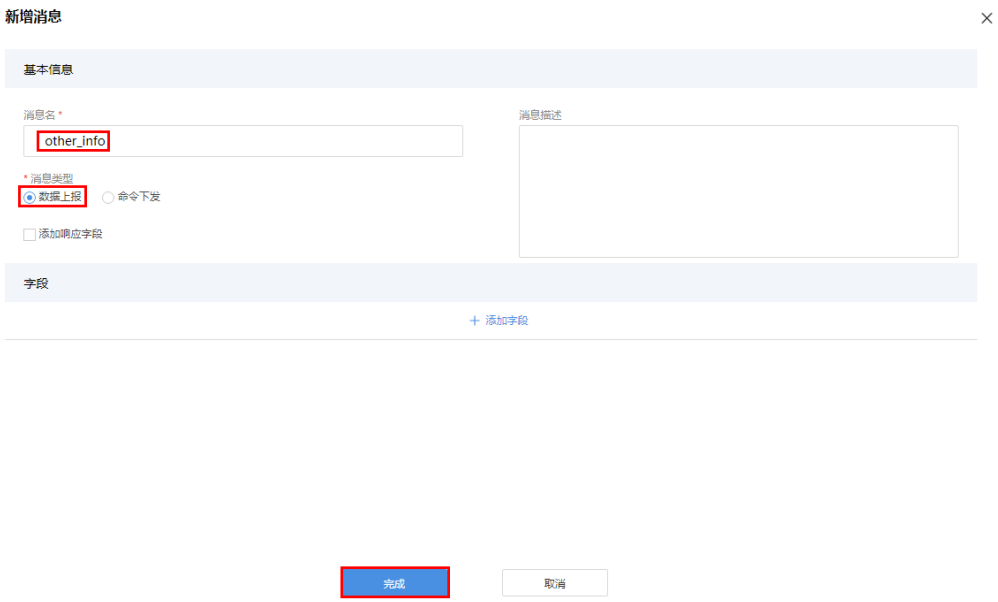
本章只讲解描述信息（other_info）上报消息的插件开发步骤，烟雾报警（level）和温度（temperature）上报消息的插件开发步骤，请参见**多条数据上报消息**。

步骤1 在烟感产品的开发空间，选择“编解码插件开发”。

步骤2 配置数据上报消息，上报火灾等级和温度，操作步骤详见**步骤2**。

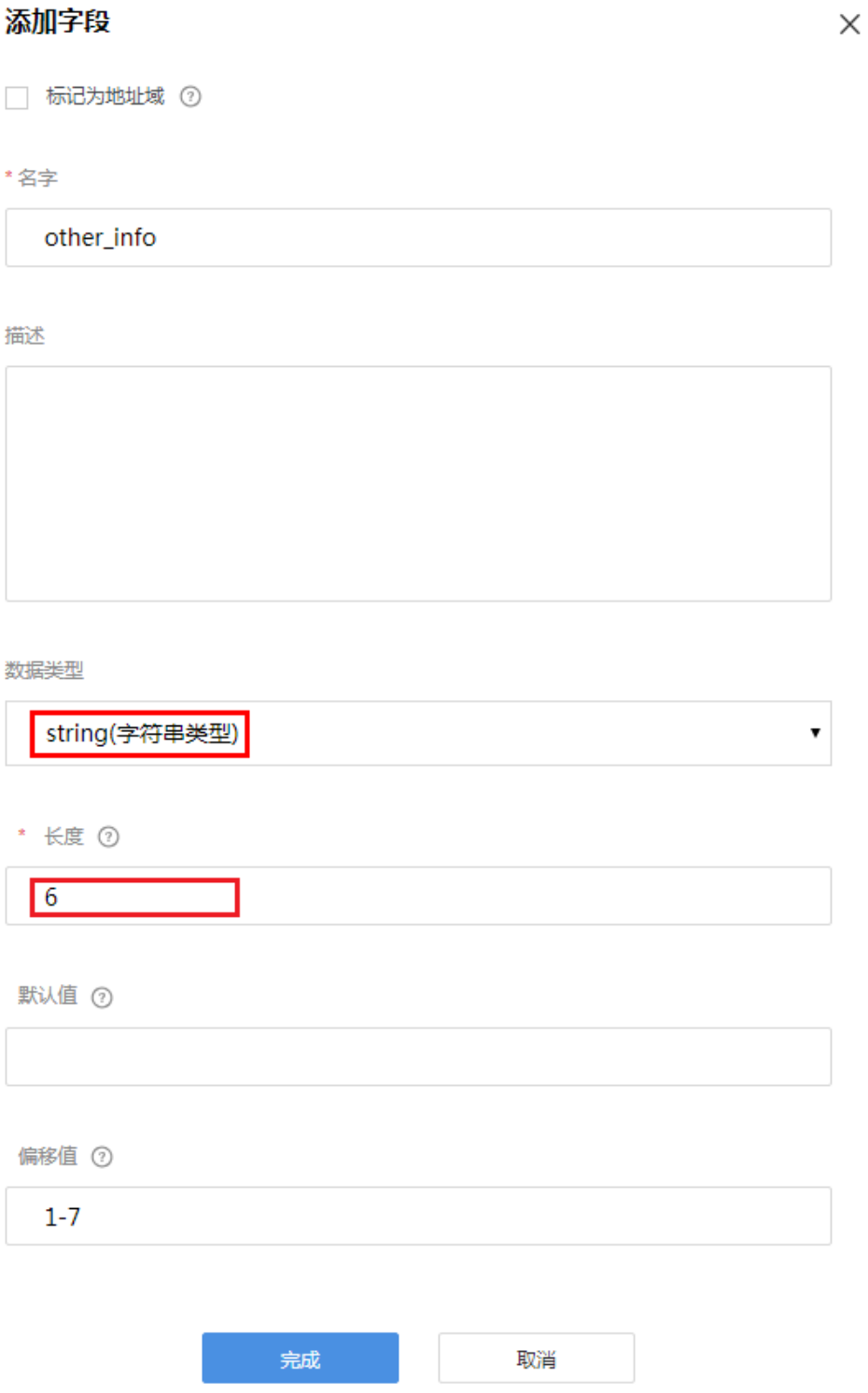
步骤3 配置数据上报消息，只上报温度，操作步骤详见**步骤3**。

步骤4 配置数据上报消息，上报字符串类型的描述信息。

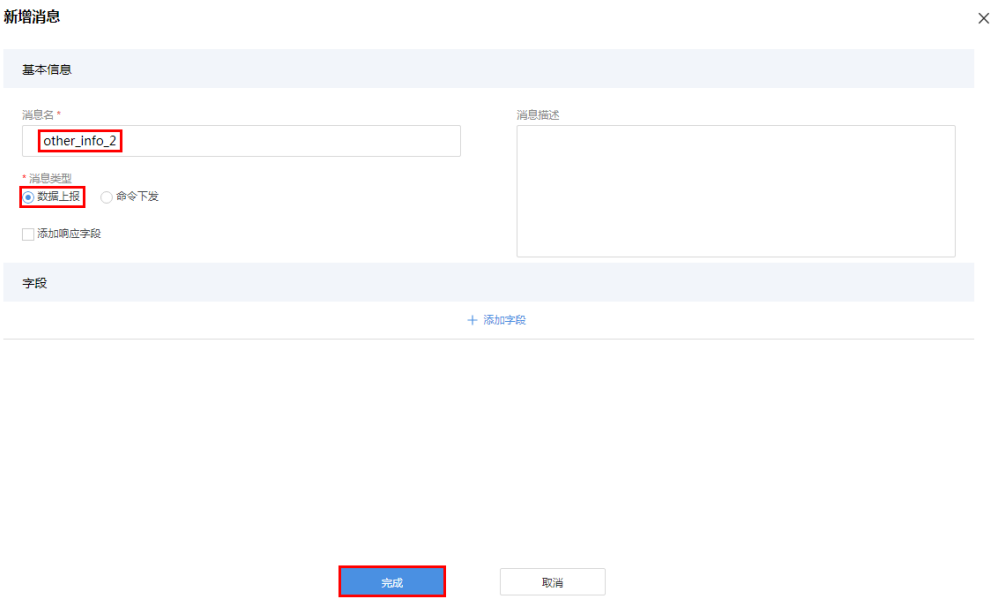


添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x2用于标识上报描述信息（字符串类型）的消息。

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。



步骤5 配置数据上报消息，上报可变长度字符串类型的描述信息。

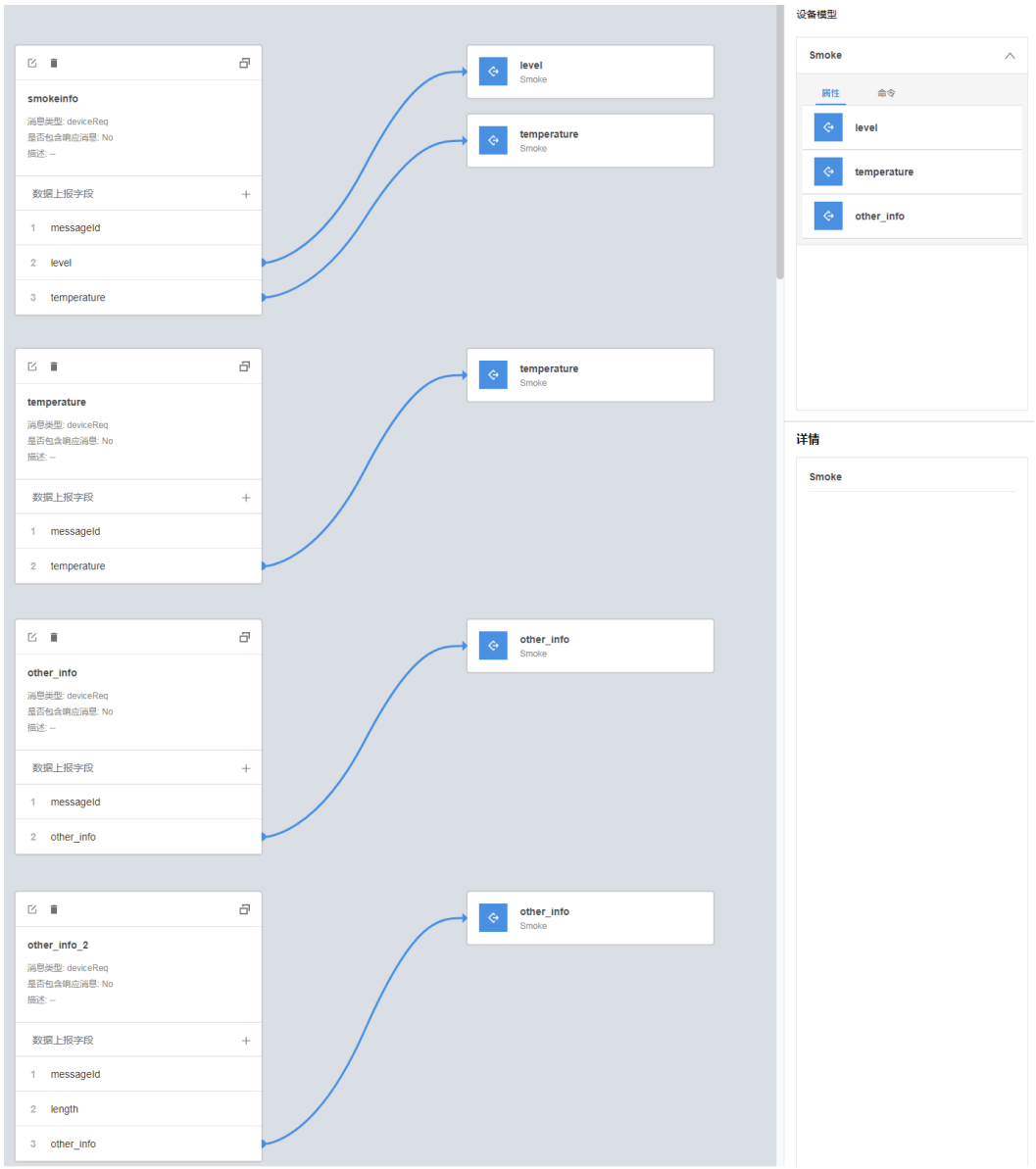


添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x3用于标识上报描述信息（可变长度字符串类型）的消息。

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。

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



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

----结束

调测编解码插件

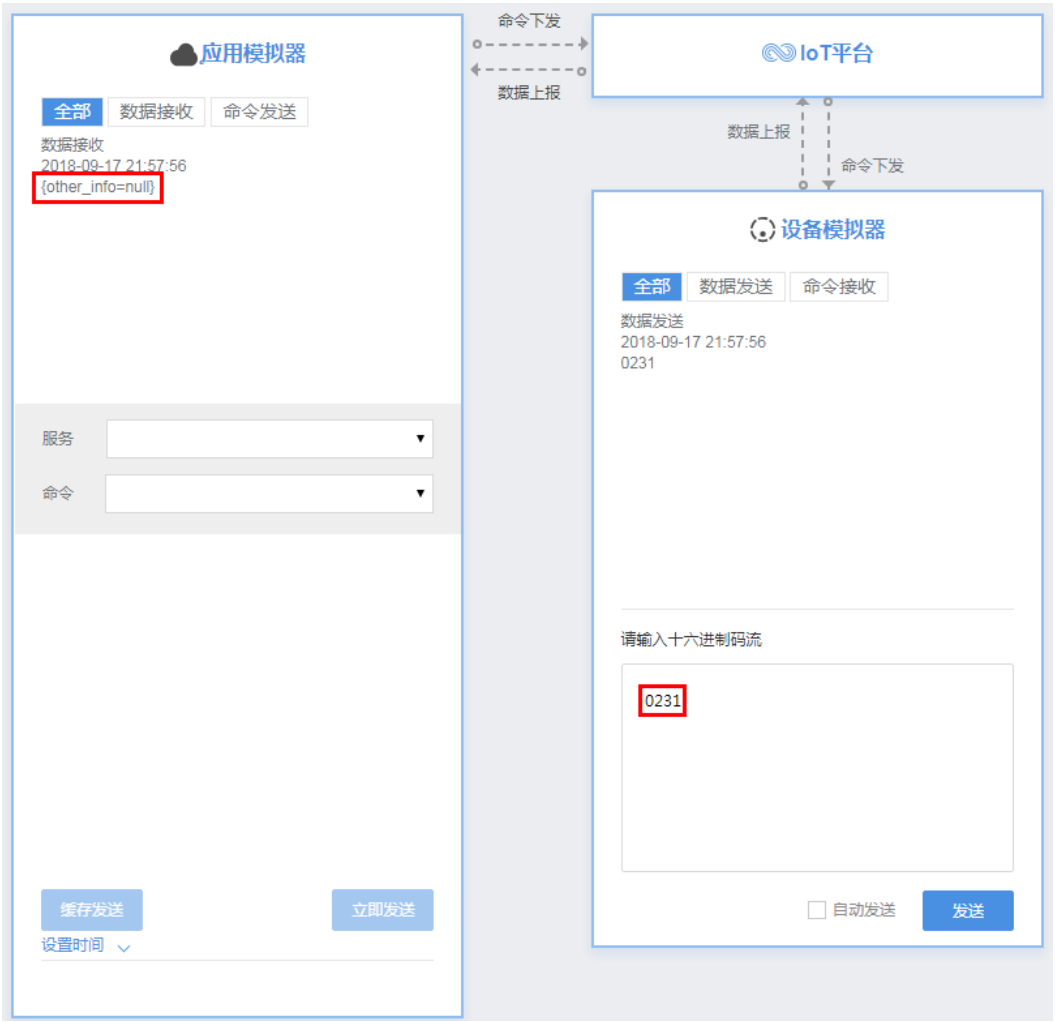
步骤1 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。

勾选“没有真实的物理设备”，点击“创建”。

步骤2 使用设备模拟器上报字符串类型的描述信息。

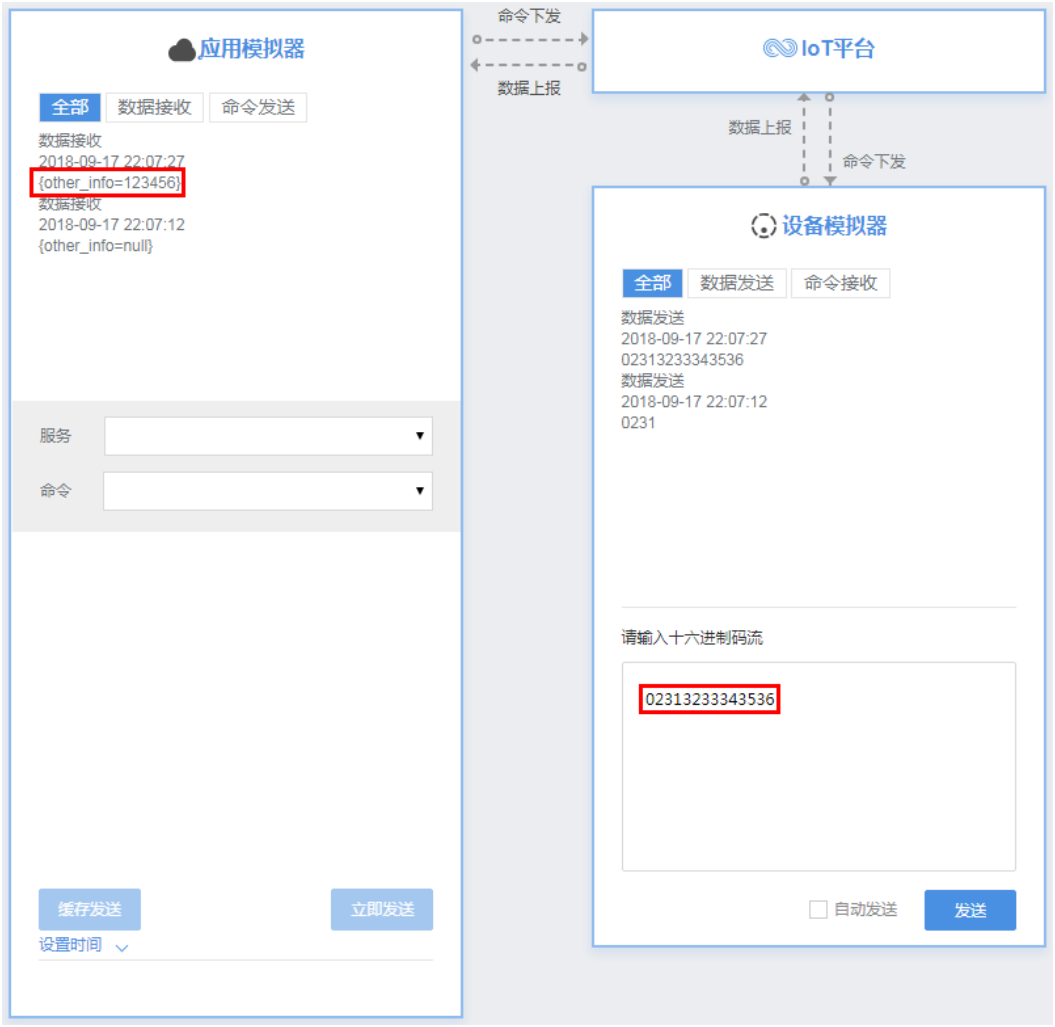
十六进制码流示例：0231。02表示messageId，此消息上报字符串类型的描述信息；31表示描述信息，长度为1个字节。

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



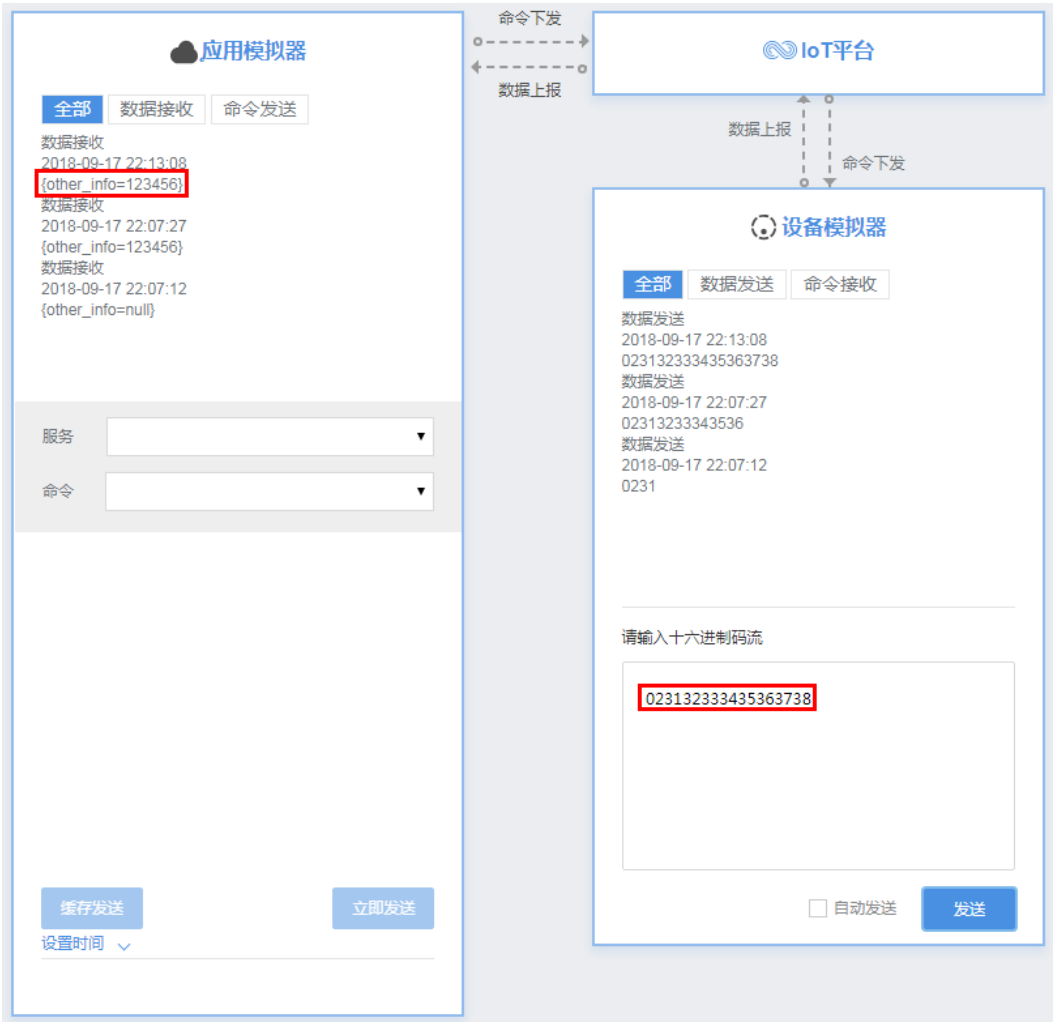
十六进制码流示例：02313233343536。02表示messageId，此消息上报字符串类型的描述信息；313233343536表示描述信息，长度为6个字节。

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



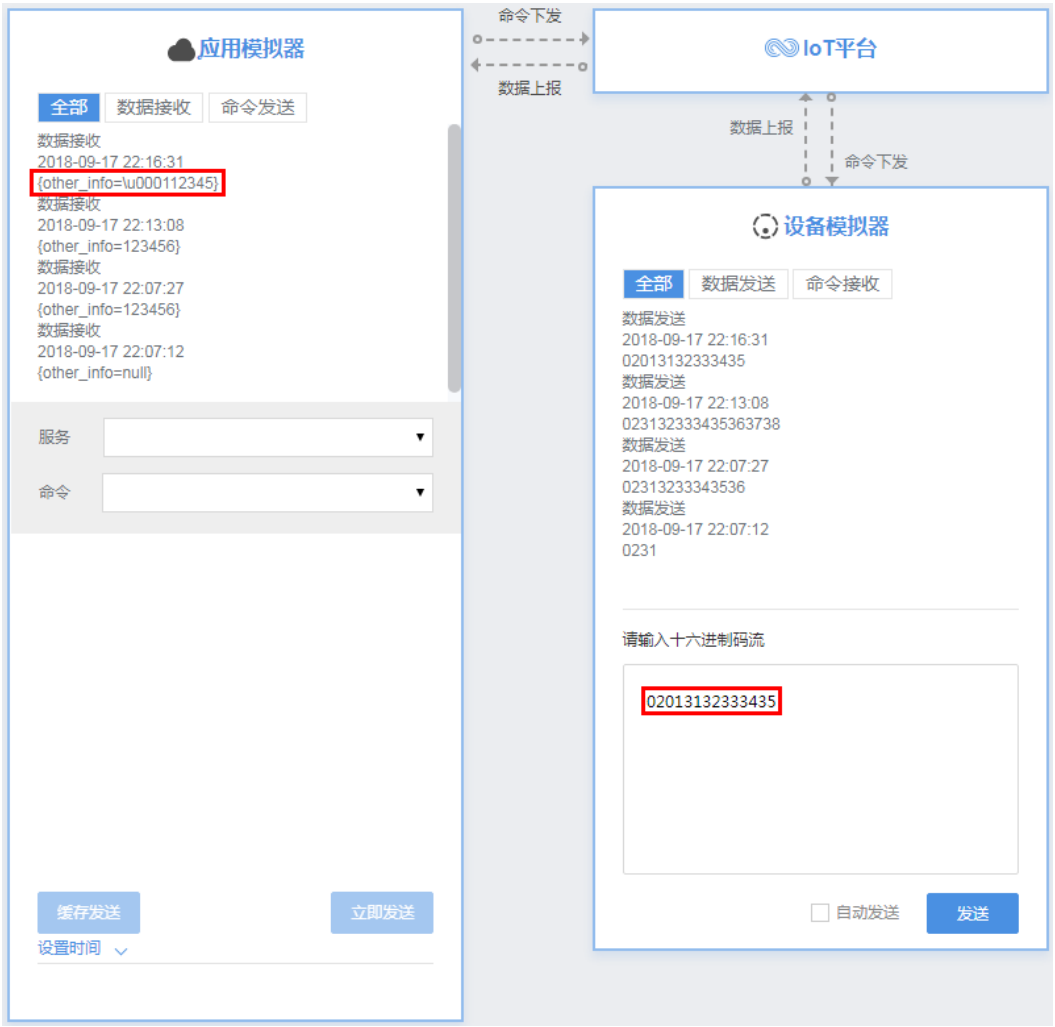
十六进制码流示例：023132333435363738。02表示messageId，此消息上报字符串类型的描述信息；3132333435363738表示描述信息，长度为8个字节。

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



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

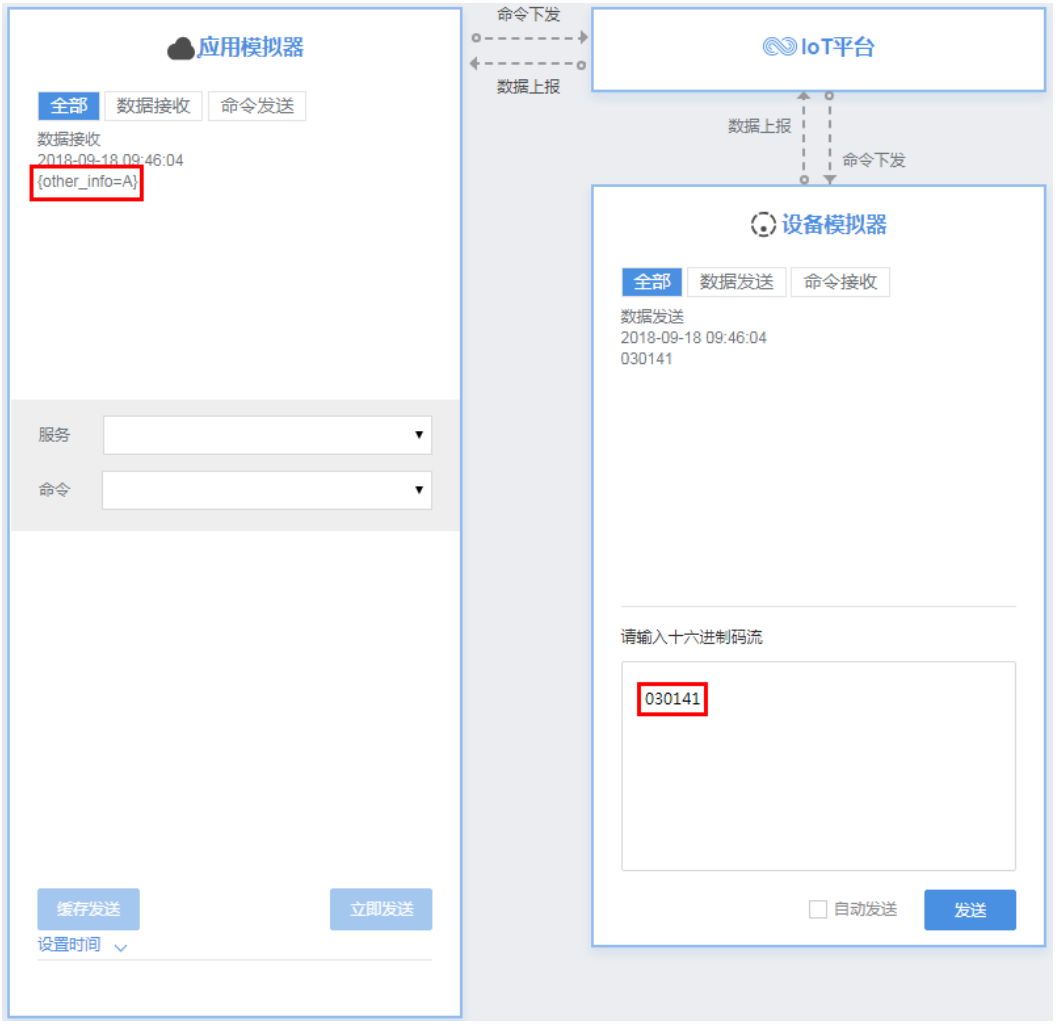
在“应用模拟器”区域查看数据上报的结果：{other_info=\u000112345}。01在ASCII码表里表示“标题开始”，无法用具体字符表示，因此编解码插件解析为\u0001。



步骤3 使用设备模拟器上报可变长度字符串类型的描述信息。

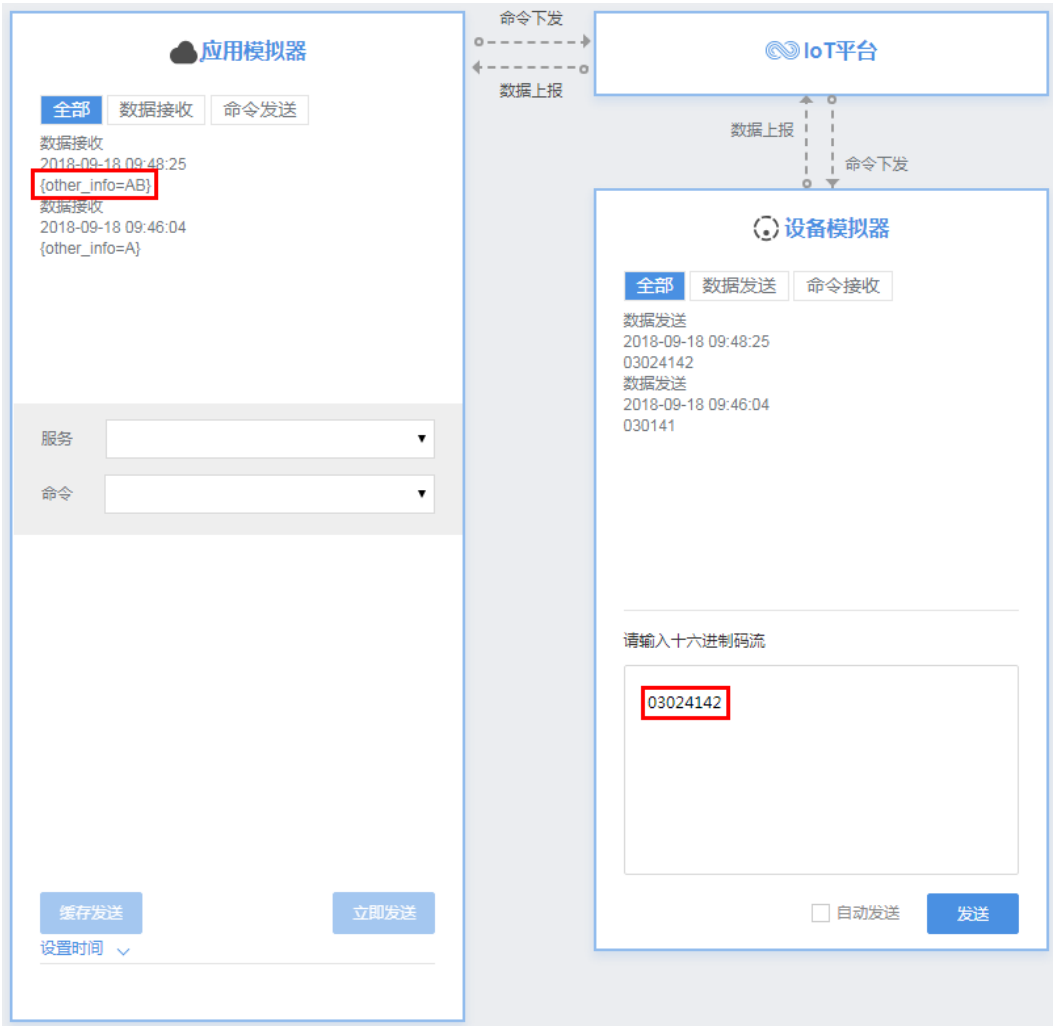
十六进制码流示例：030141。03表示messageId，此消息上报可变长度字符串类型的描述信息；01表示描述信息长度（1个字节），长度为1个字节；41表示描述信息，长度为1个字节。

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



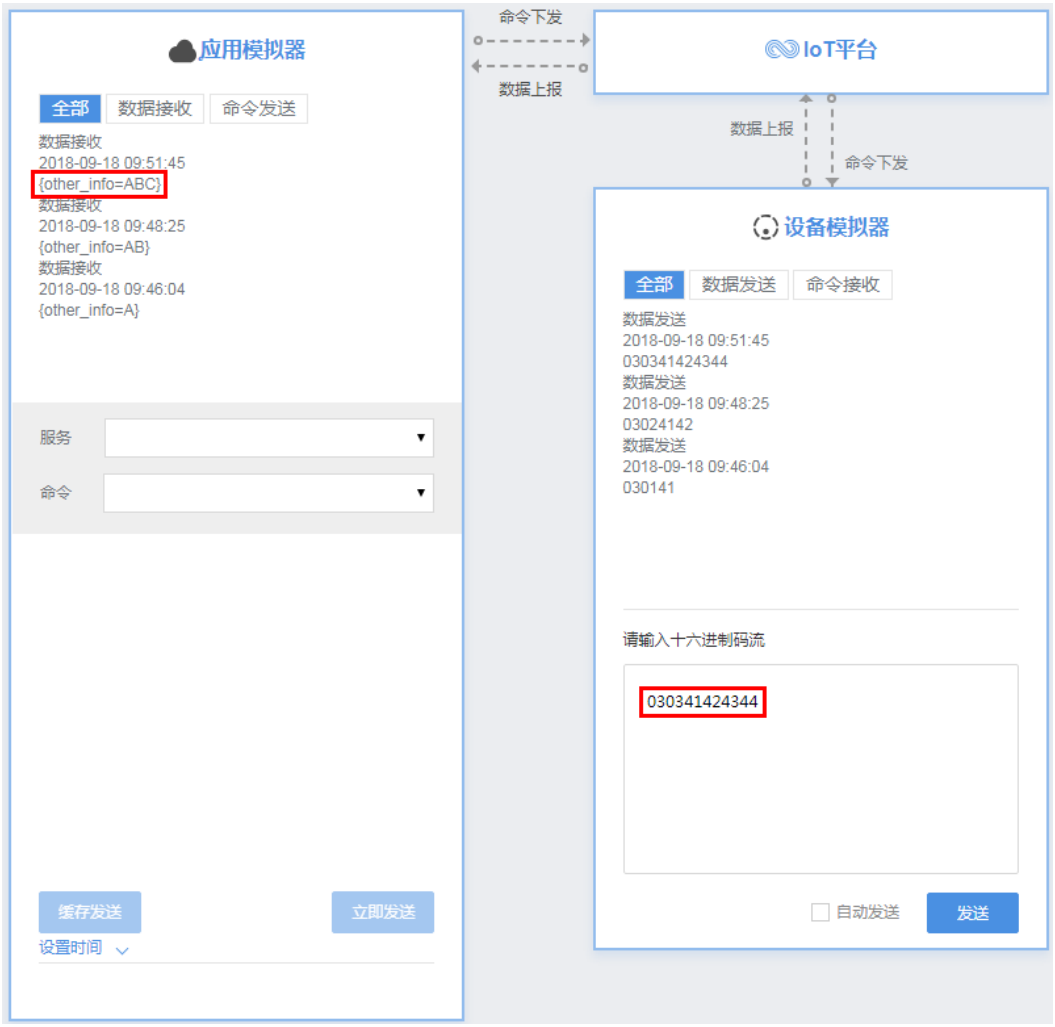
十六进制码流示例：03024142。03表示messageId，此消息上报可变长度字符串类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；4142表示描述信息，长度为2个字节。

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



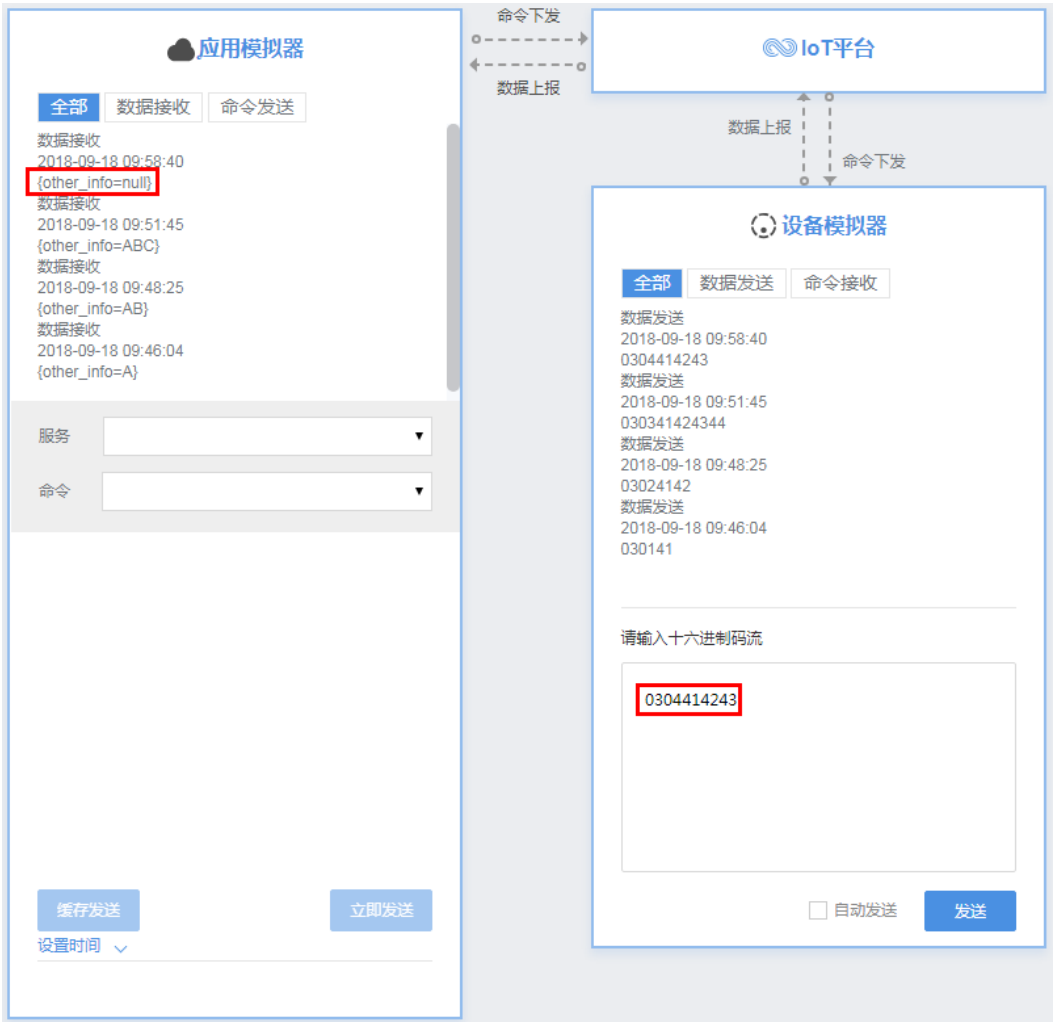
十六进制码流示例：030341424344。03表示messageId，此消息上报可变长度字符串类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；41424344表示描述信息，长度为4个字节。

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



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

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



----结束

总结

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

1.4.3.4 数组及可变长数组数据类型

场景说明

有一款烟感设备，具有如下特征：

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

说明

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

Profile 定义

在烟感产品的开发空间完成Profile定义。

编解码插件开发

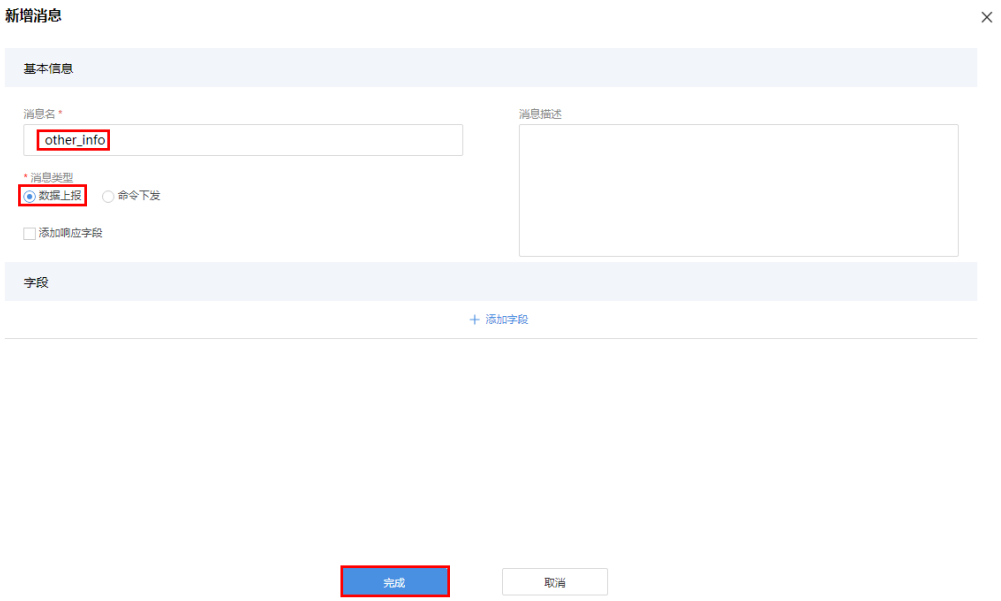
本章只讲解描述信息（other_info）上报消息的插件开发步骤，烟雾报警（level）和温度（temperature）上报消息的插件开发步骤，请参见**多条数据上报消息**。

步骤1 在烟感产品的开发空间，选择“编解码插件开发”。

步骤2 配置数据上报消息，上报火灾等级和温度，操作步骤详见**步骤2**。

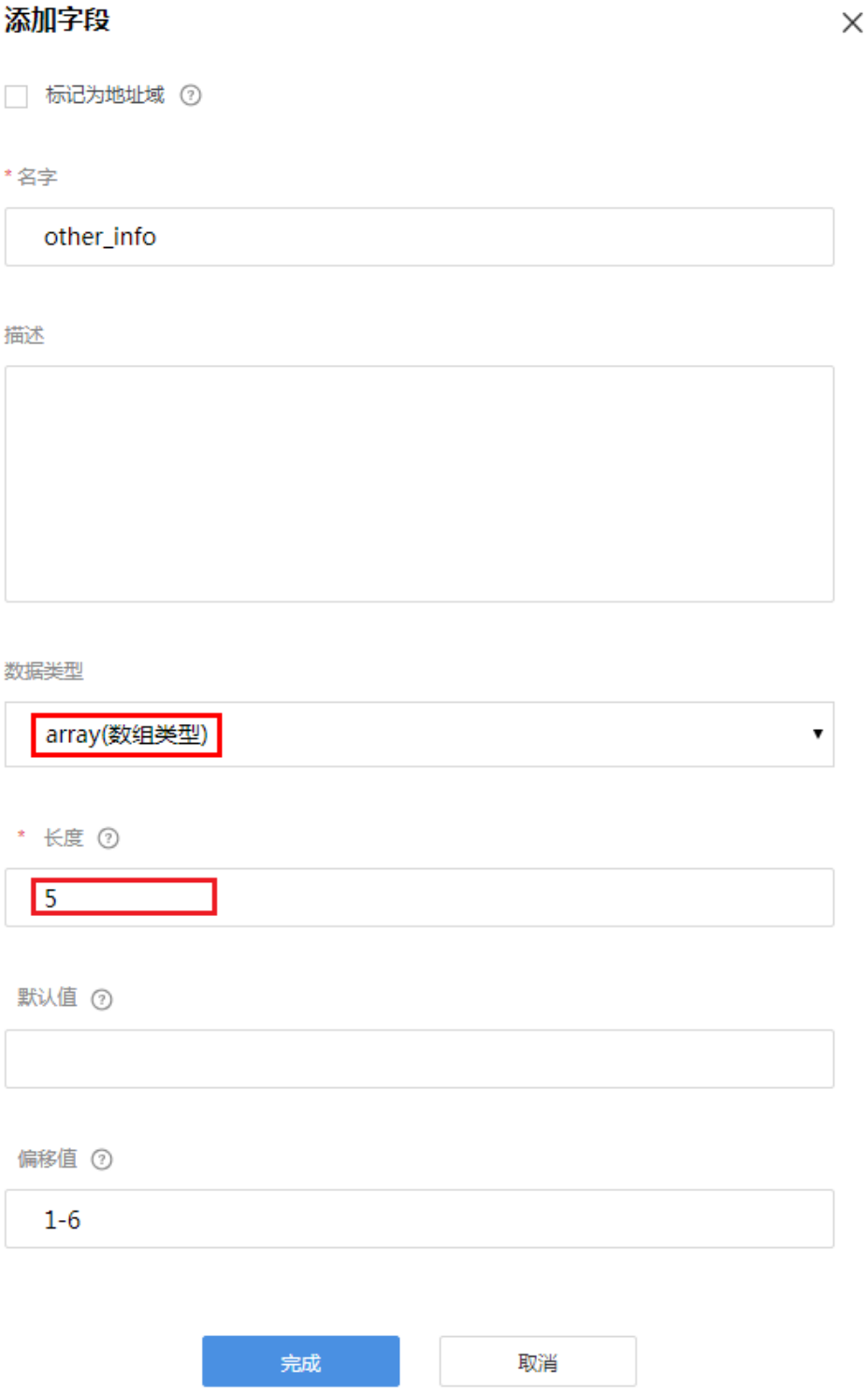
步骤3 配置数据上报消息，只上报温度，操作步骤详见**步骤3**。

步骤4 配置数据上报消息，上报数组类型的描述信息。

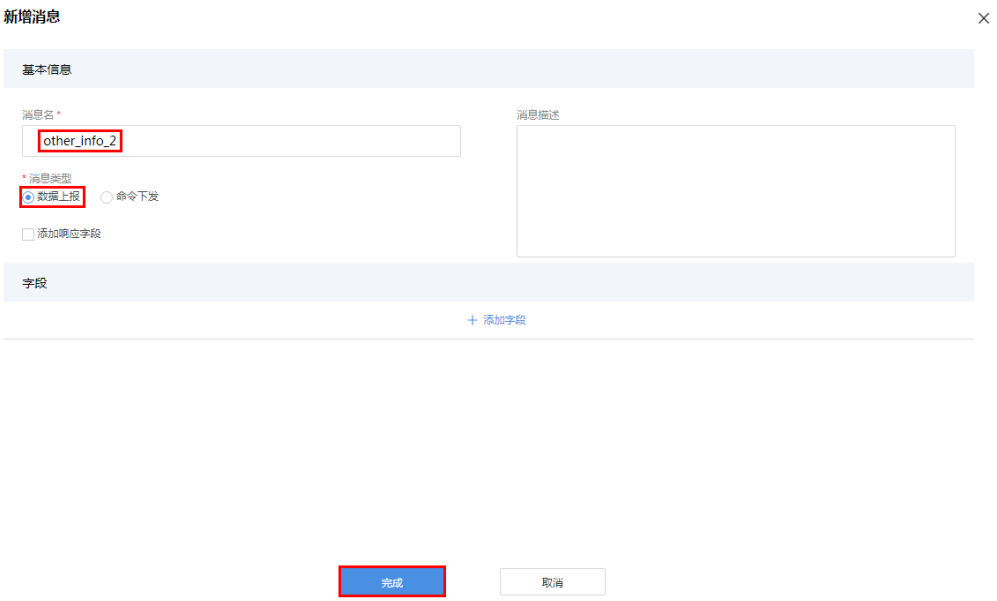


添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x2用于标识上报描述信息（数组类型）的消息。

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。



步骤5 配置数据上报消息，上报可变长度数组类型的描述信息。

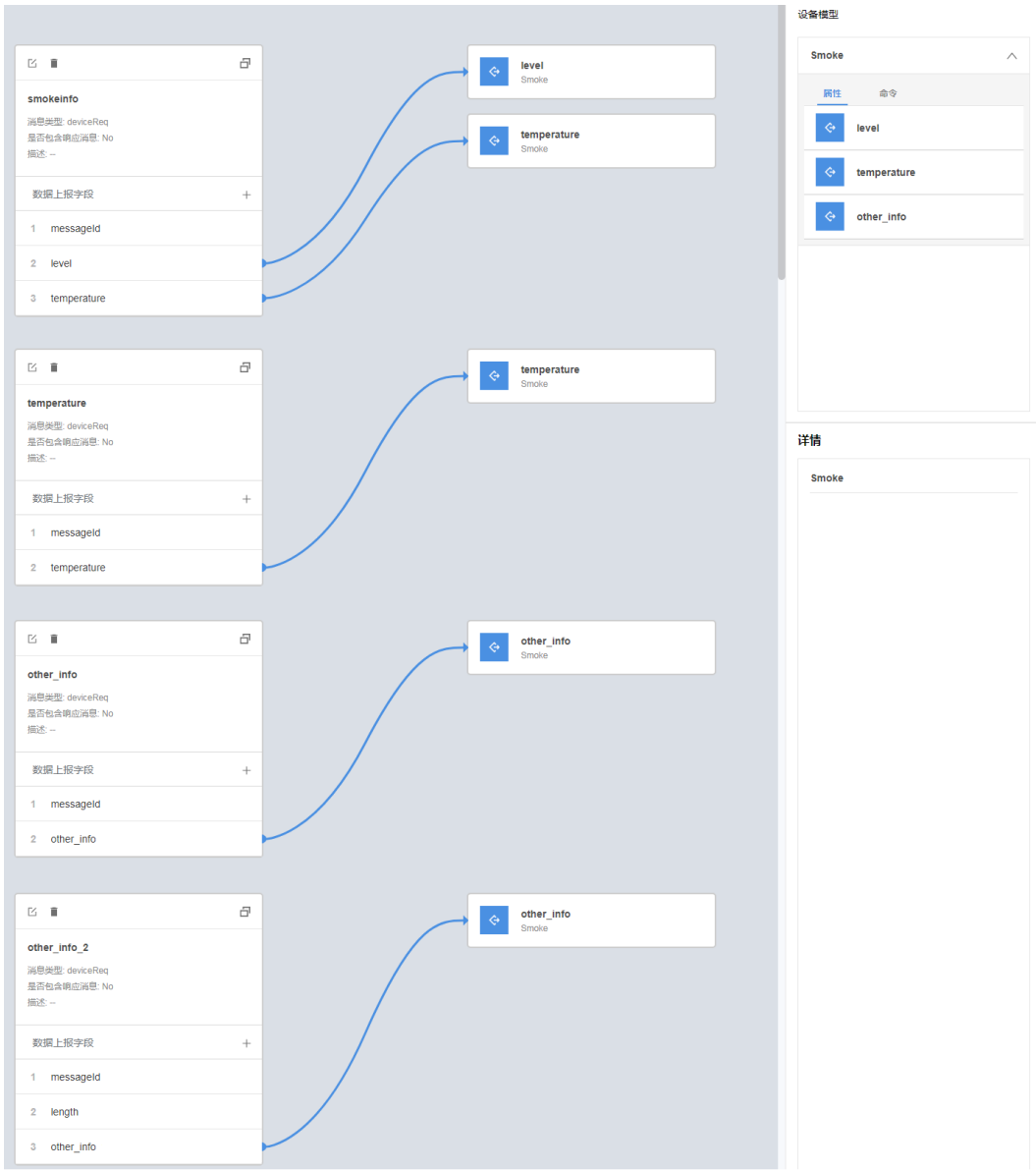


添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x3用于标识上报描述信息（可变长度数组类型）的消息。

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。

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



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

----结束

调测编解码插件

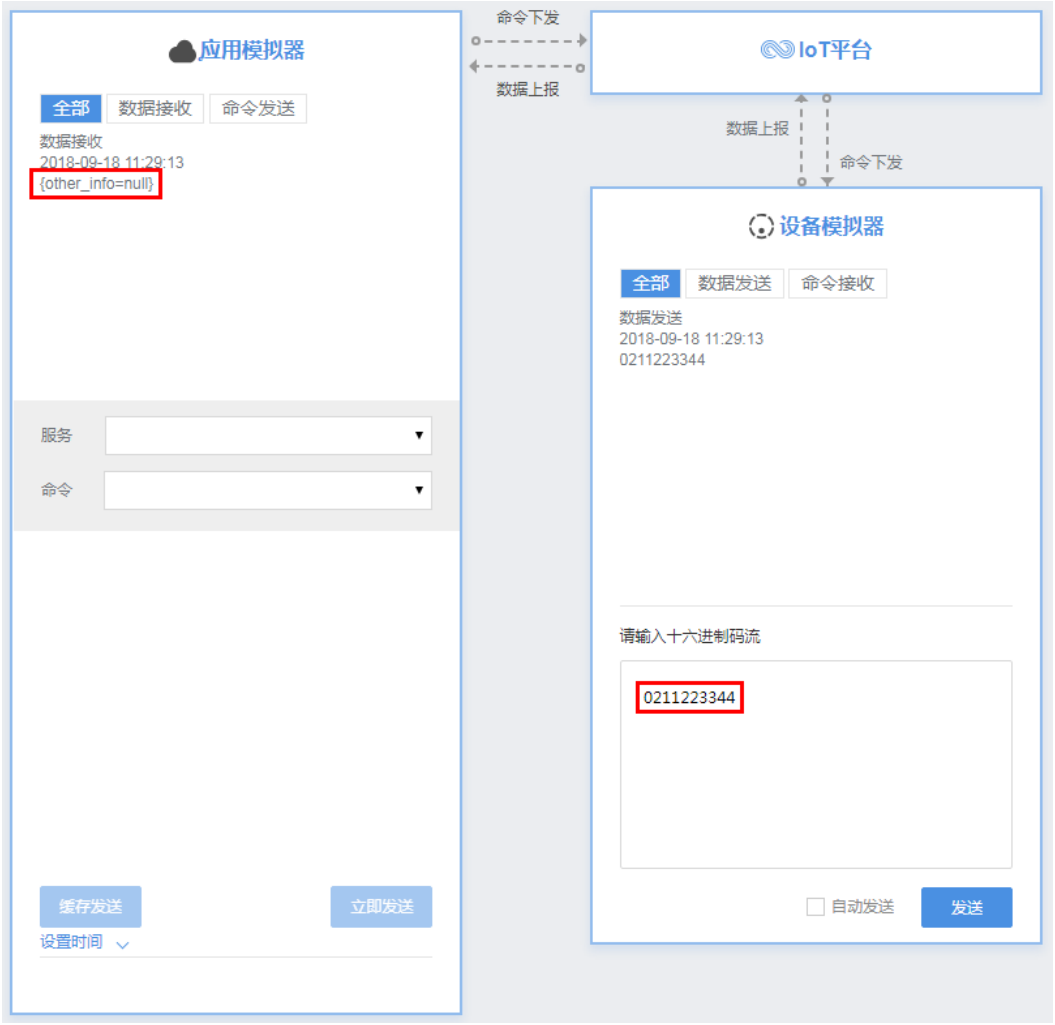
步骤1 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。

勾选“没有真实的物理设备”，点击“创建”。

步骤2 使用设备模拟器上报数组类型的描述信息。

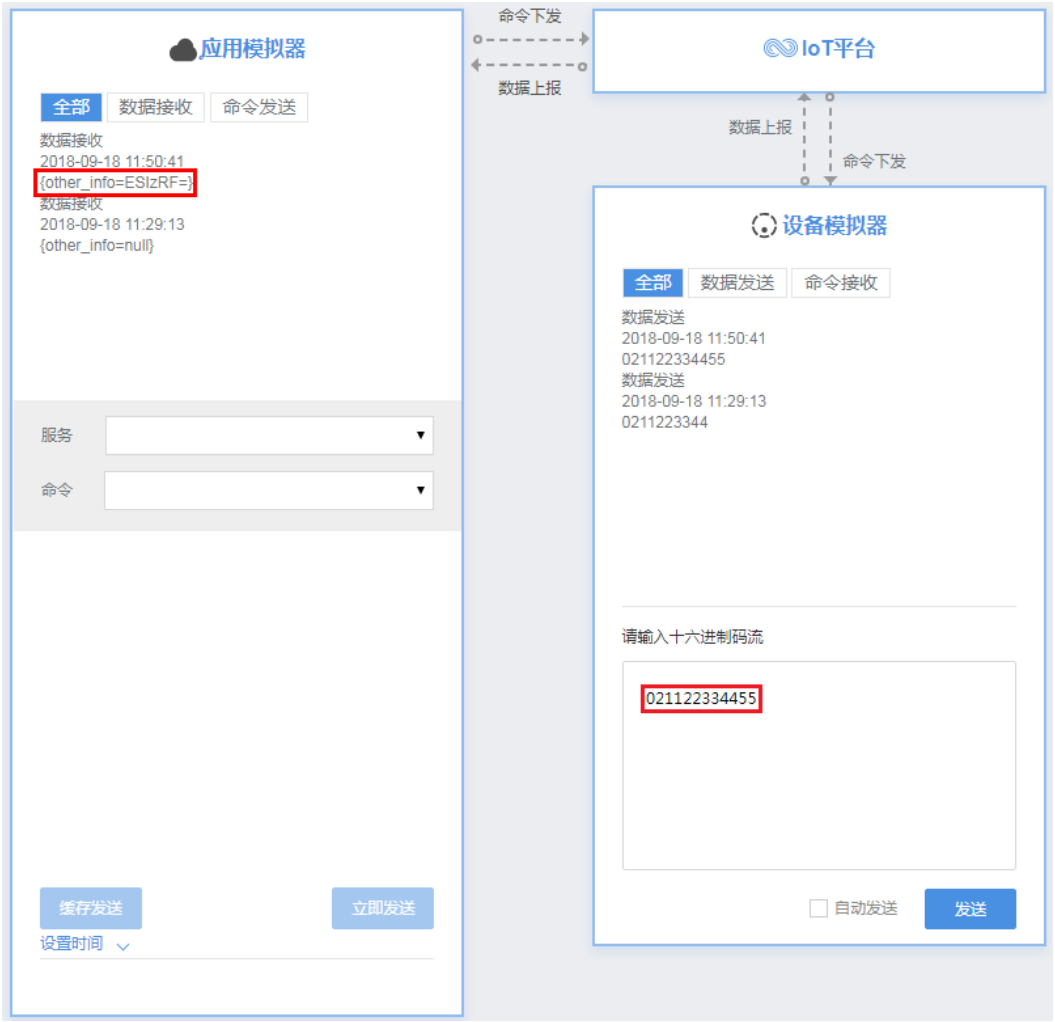
十六进制码流示例：0211223344。02表示messageId，此消息上报数组类型的描述信息；11223344表示描述信息，长度为4个字节。

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



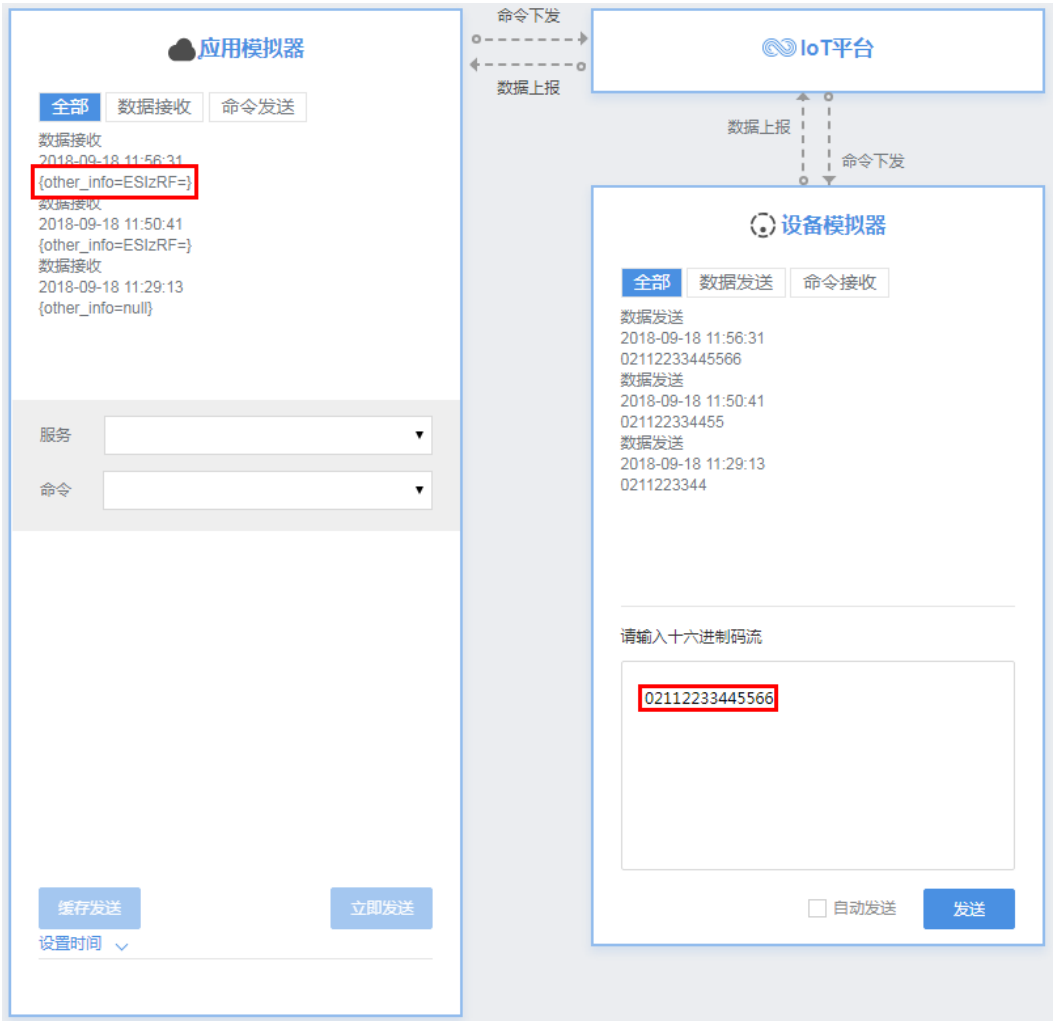
十六进制码流示例：021122334455。02表示messageId，此消息上报数组类型的描述信息；1122334455表示描述信息，长度为5个字节。

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



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

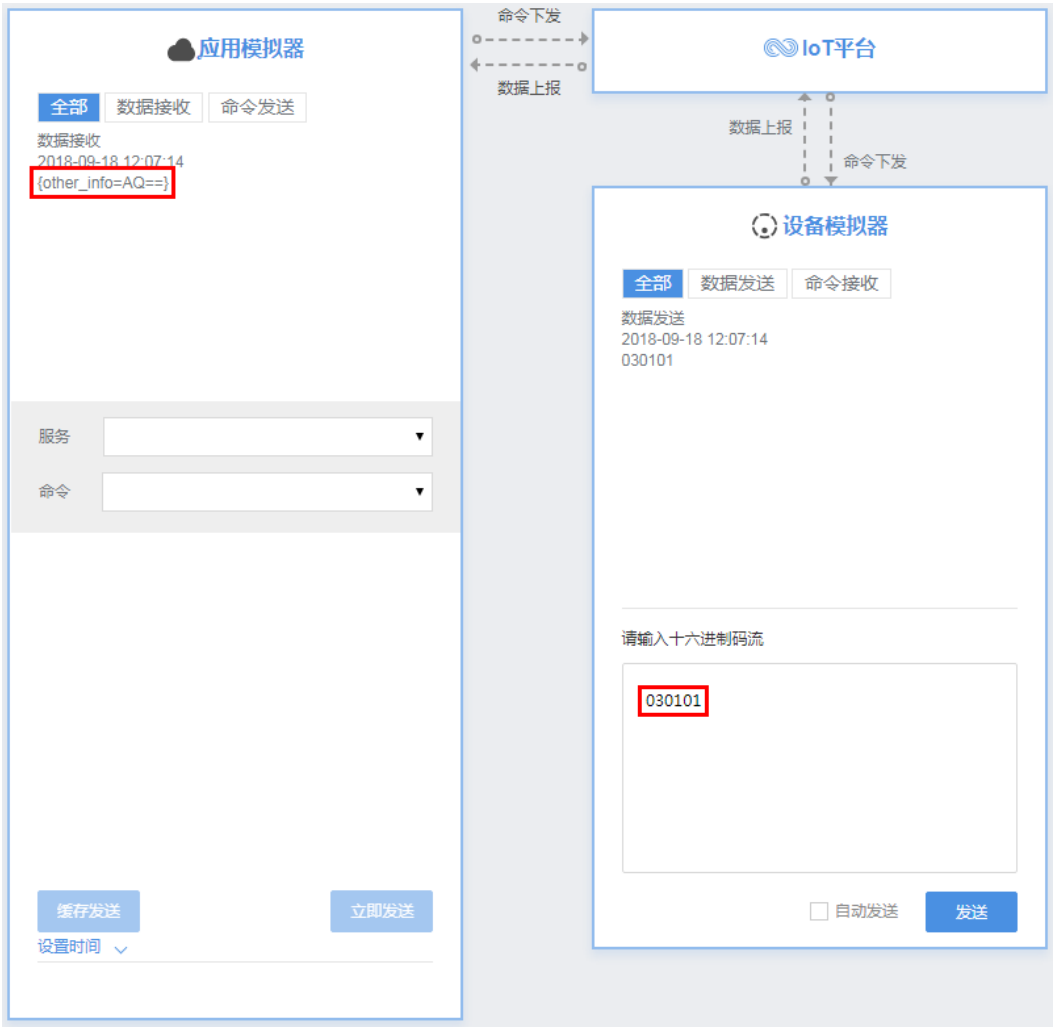
在“应用模拟器”区域查看数据上报的结果：{other_info=ESIzRF=}。描述信息长度超过5个字节，编解码插件截取前5个字节进行解析。



步骤3 使用设备模拟器上报可变长度数组类型的描述信息。

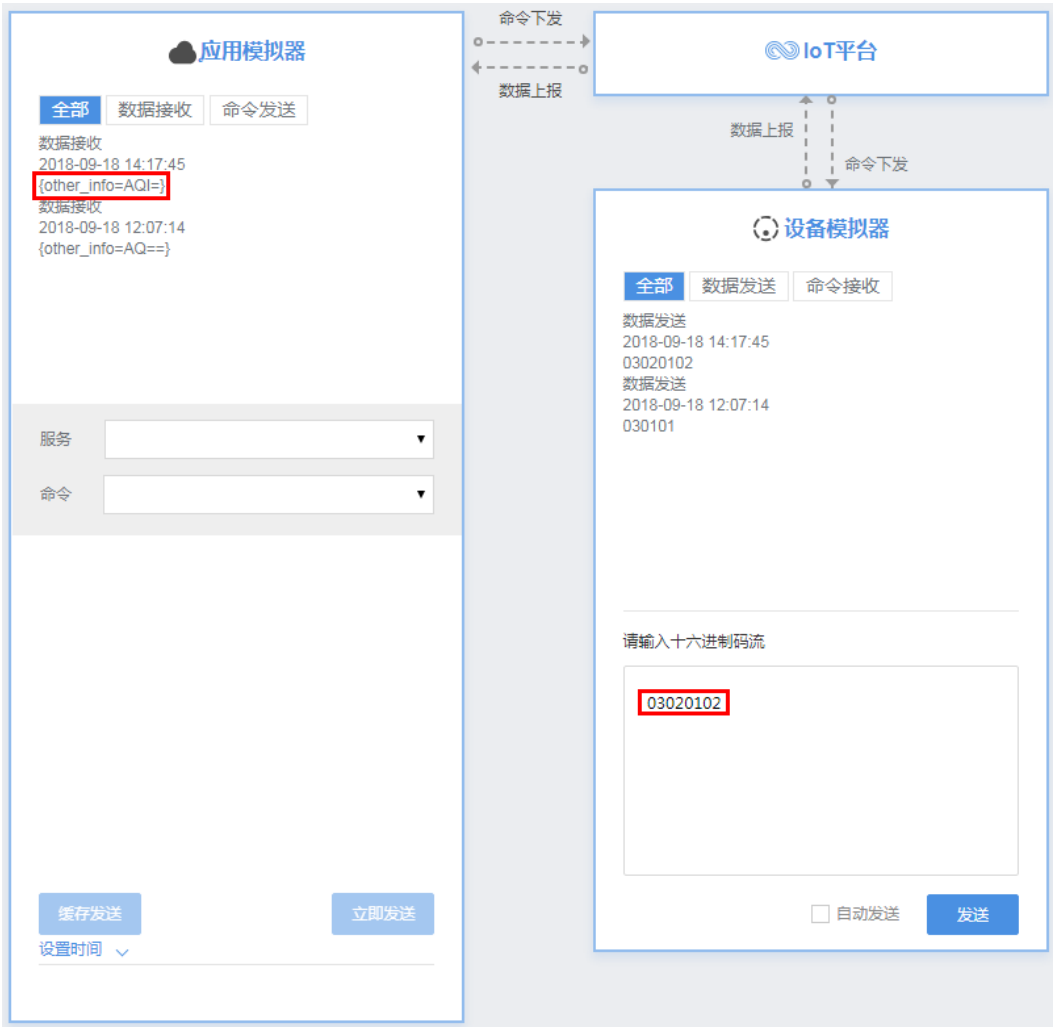
十六进制码流示例：030101。03表示messageId，此消息上报可变长度数组类型的描述信息；01表示描述信息长度（1个字节），长度为1个字节；01表示描述信息，长度为1个字节。

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



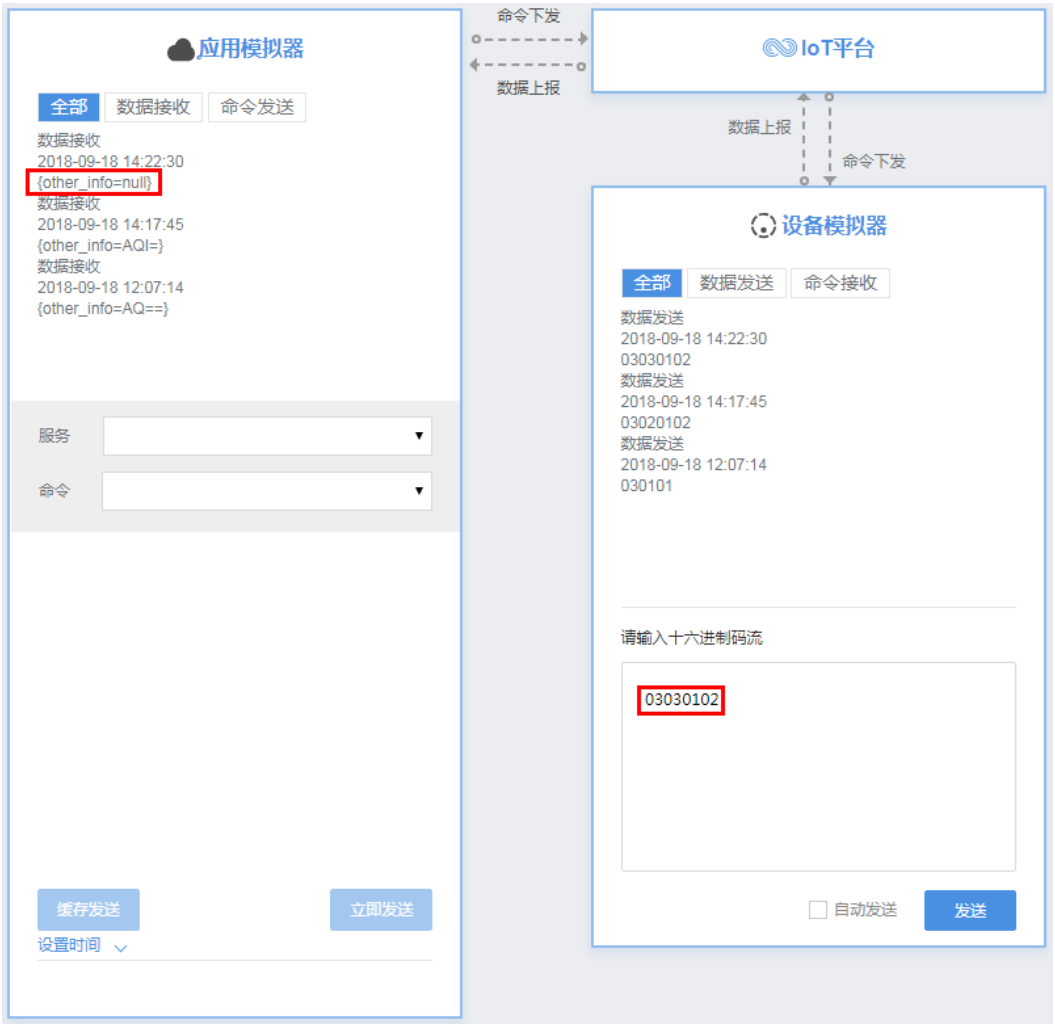
十六进制码流示例：03020102。03表示messageId，此消息上报可变长度数组类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

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



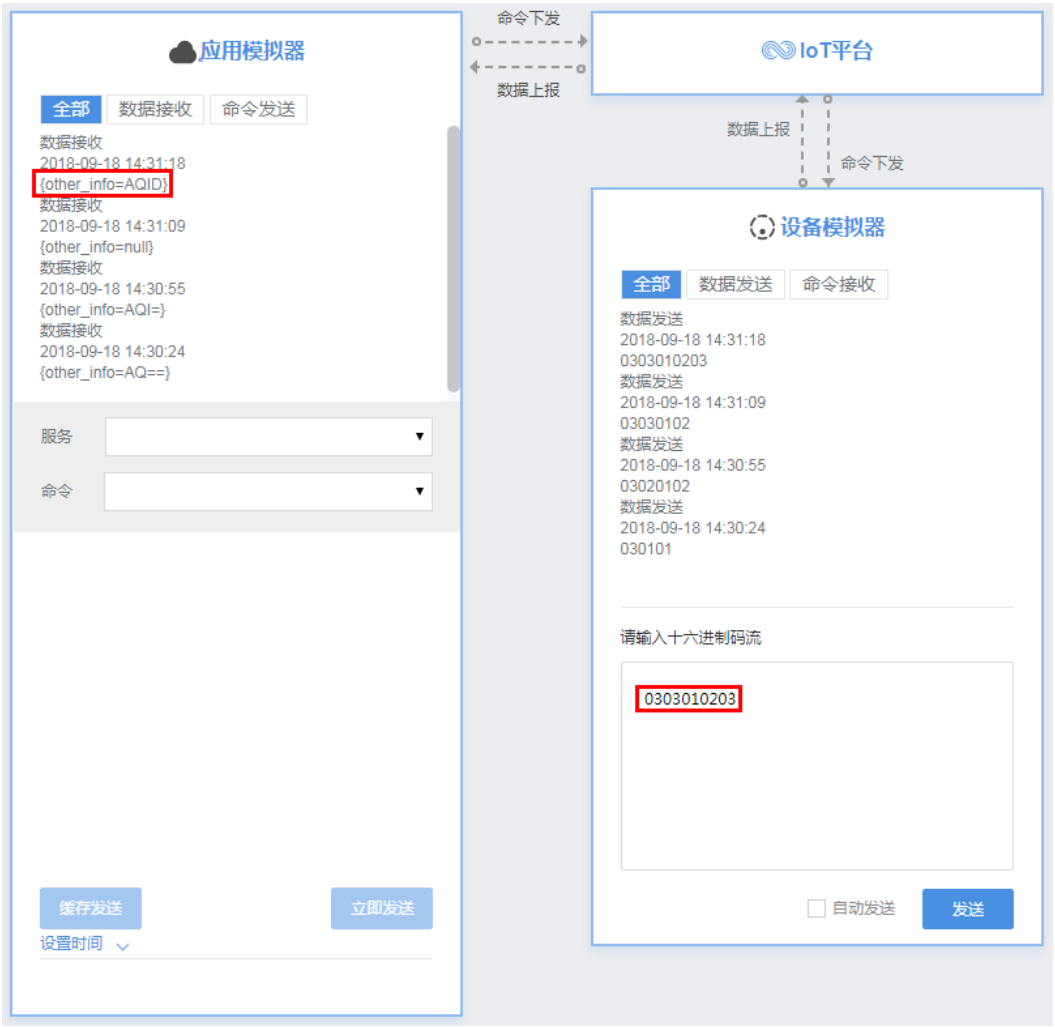
十六进制码流示例：03030102。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

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



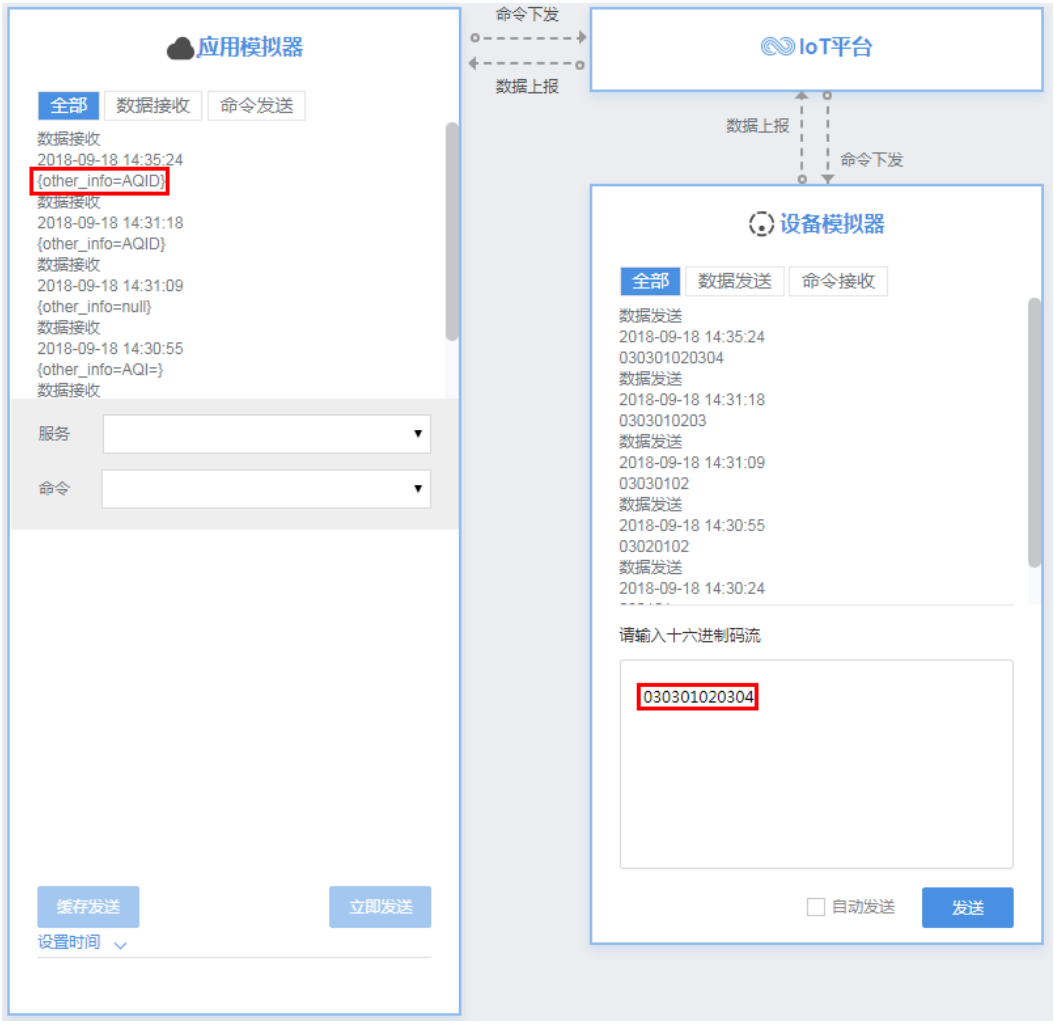
十六进制码流示例：0303010203。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；010203表示描述信息，长度为3个字节。

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



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

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



----结束

base64 编码方式说明

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

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

- 把01当作字符，不足3个字符，补1个0，得到010。通过查询ASCII码表，将字符转换为8位二进制数，即：0转换为00110000、1转换为00110001，因此010可以转换为001100000011000100110000（3*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*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。

1.4.3.5 含命令执行结果的编解码插件

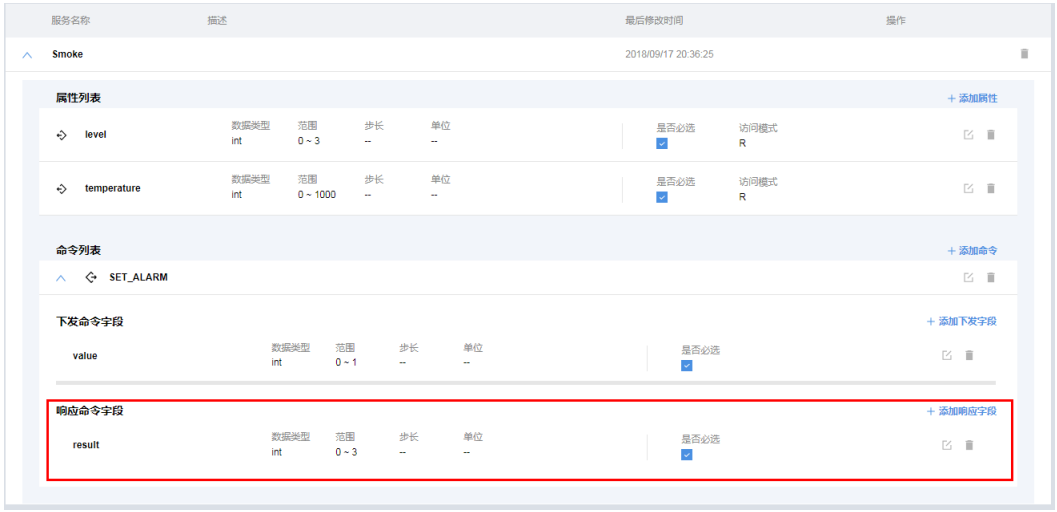
场景说明

有一款烟感设备，具有如下特征：

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

Profile 定义

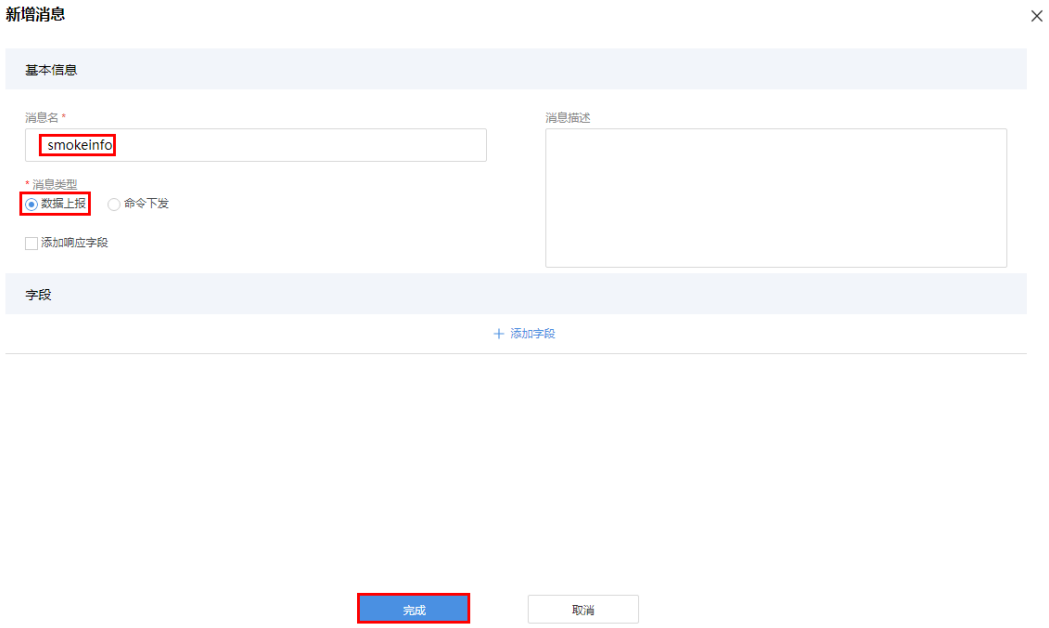
在烟感产品的开发空间完成Profile定义。



编解码插件开发

步骤1 在烟感产品的开发空间，选择“编解码插件开发”。

步骤2 配置数据上报消息，上报火灾等级和温度。



添加messageId字段，表示消息种类。

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

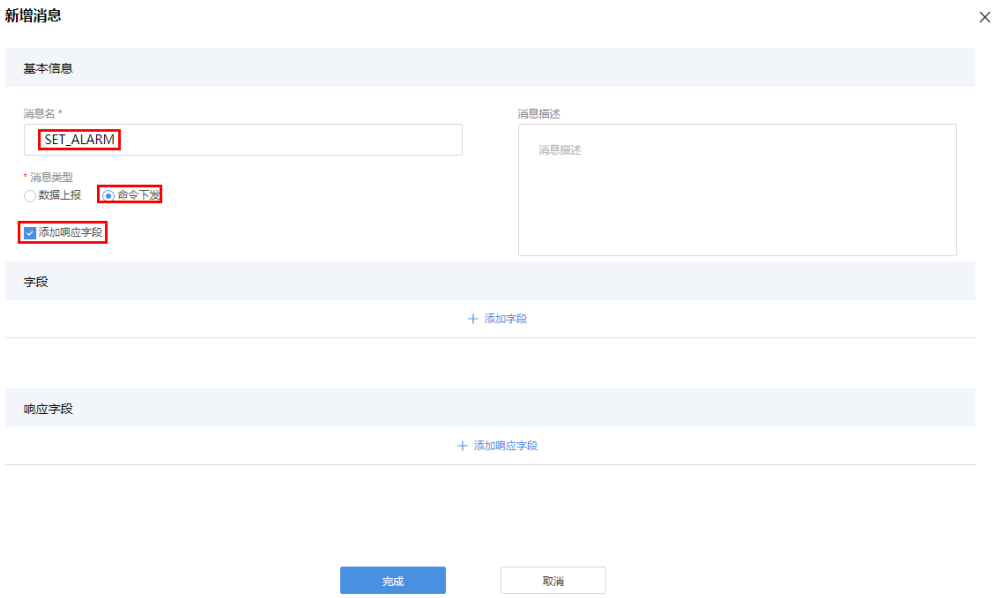
添加level字段，表示火灾级别。

- “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。

步骤3 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mid字段，用于将下发的命令和命令执行结果进行关联。

添加value字段，表示下发命令的参数值。

步骤4 配置命令下发响应消息。

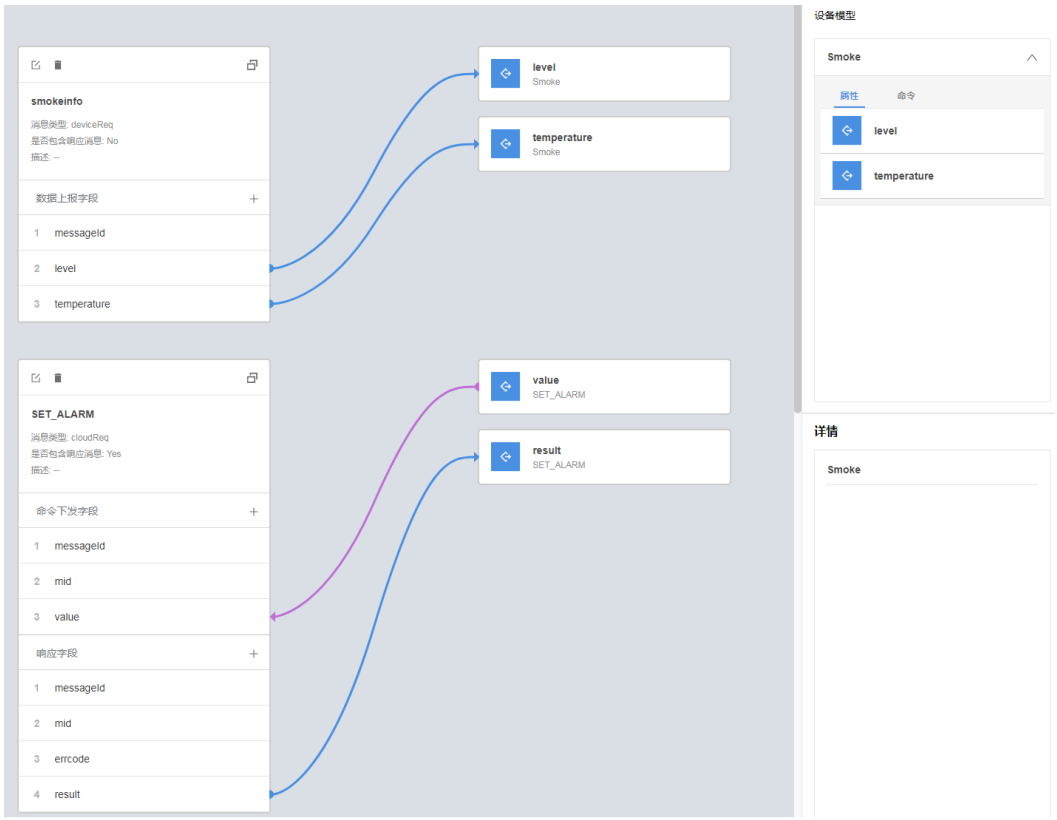
添加messageId，表示消息种类。命令执行结果为上行消息，需要通过messageId和数据上报消息进行区分。

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result字段，用于表示命令执行结果。

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



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

----结束

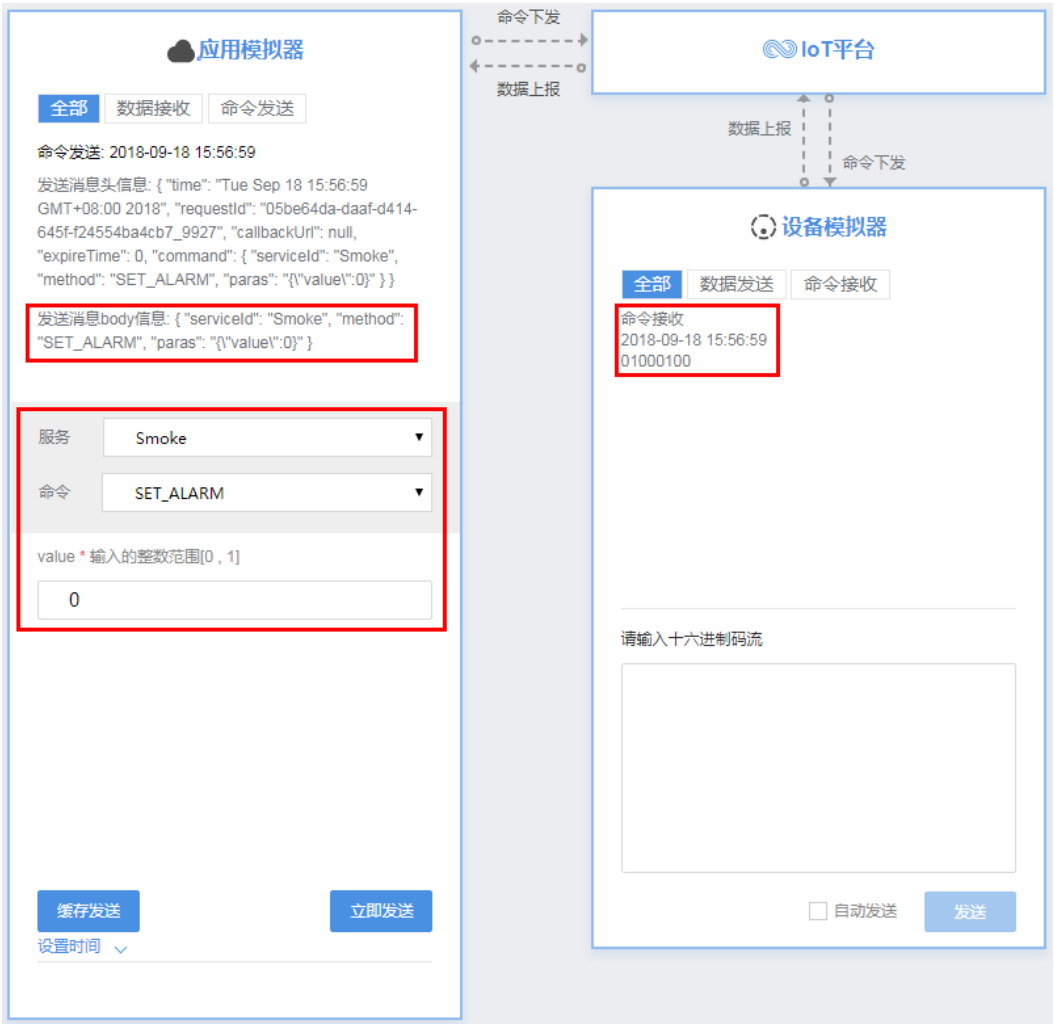
调测编解码插件

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

勾选“没有真实的物理设备”，点击“创建”。

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

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



步骤3 使用设备模拟器进行数据上报。

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

在“设备详情 > 历史命令”查看命令执行状态：执行成功。

----结束

总结

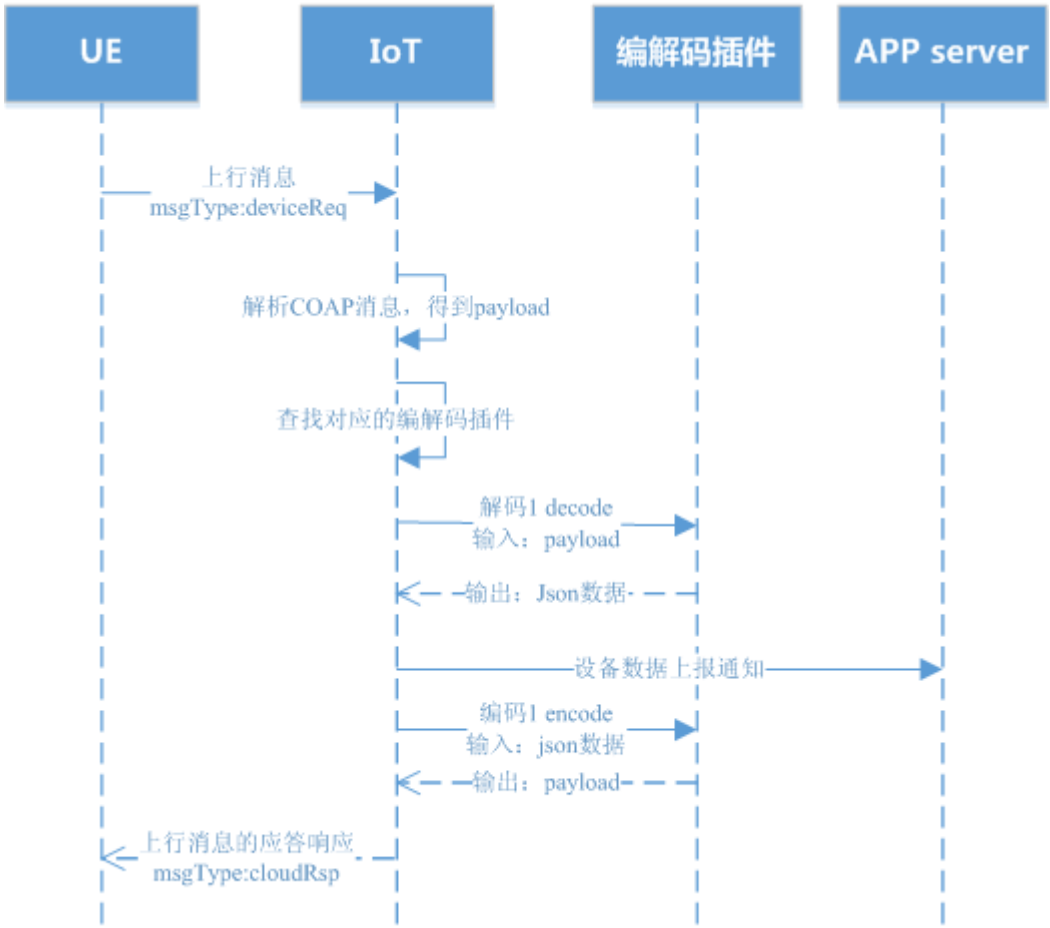
- 如果插件需要对命令执行结果进行解析，则必须在命令和命令响应中定义mid字段。
- 命令下发的mid是2个字节，对于每个设备来说，mid从1递增到65535，对应码流为0001到FFFF。
- 设备执行完命令，命令执行结果上报中的mid要与收到命令中的mid保持一致，这样平台才能刷新对应命令的状态。

1.4.4 参考信息

1.4.4.1 消息处理流程

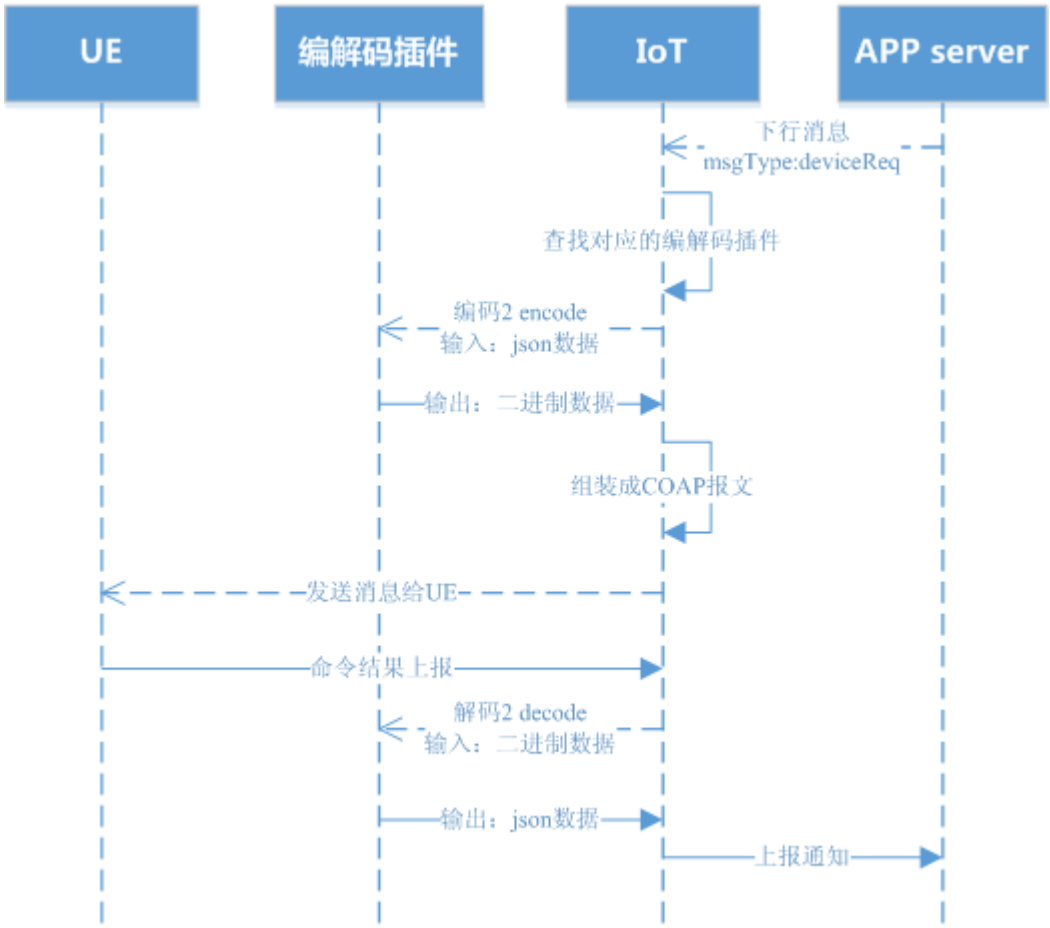
数据上报

图 1-19 数据上报处理流程



命令下发处理

图 1-20 命令下发处理流程



1.4.4.2 decode 接口说明

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

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

图 1-21 上行报文

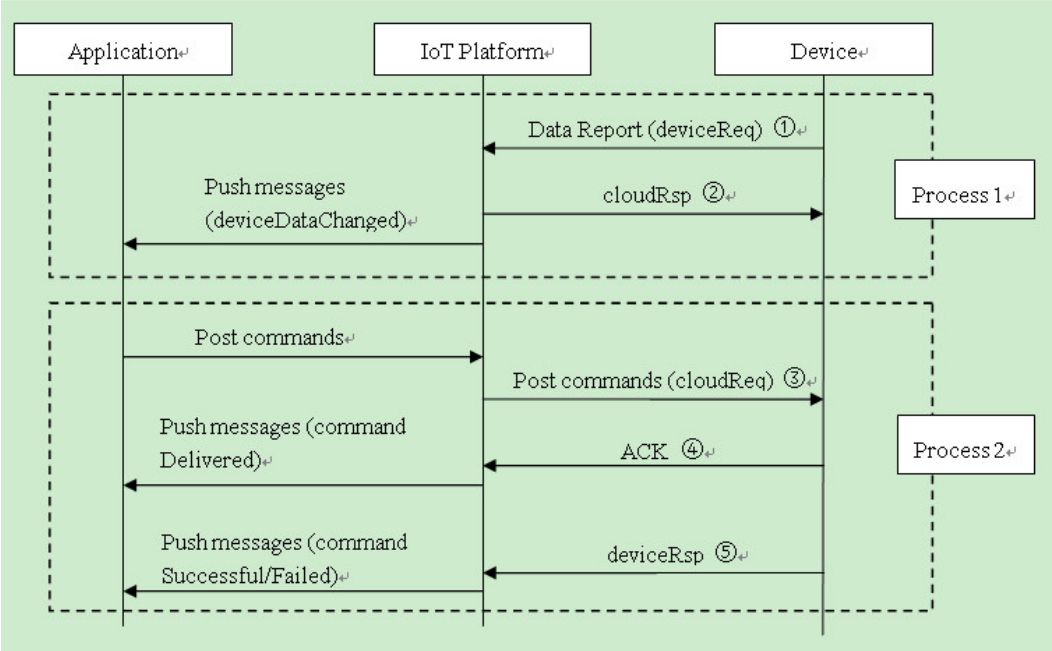


表 1-4 设备上报数据

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

表 1-5 ArrayNode 定义

字段名	类型	参数描述	是否必填
serviceId	String	服务的id。	是

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

示例：

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

表 1-6 设备对平台命令的应答

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

字段名	类型	参数描述	是否必填
errcode	Int	请求处理的结果码，IoT平台根据该参数判断命令下发的状态。 0表示成功，1表示失败。	是
body	ObjectNode	命令的应答，具体字段由profile定义。 **说明** body体不是数组。	否

示例：

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

1.4.4.3 encode 接口说明

encode接口的入参json格式数据，是平台下发的命令或应答。

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

图 1-22 下行报文

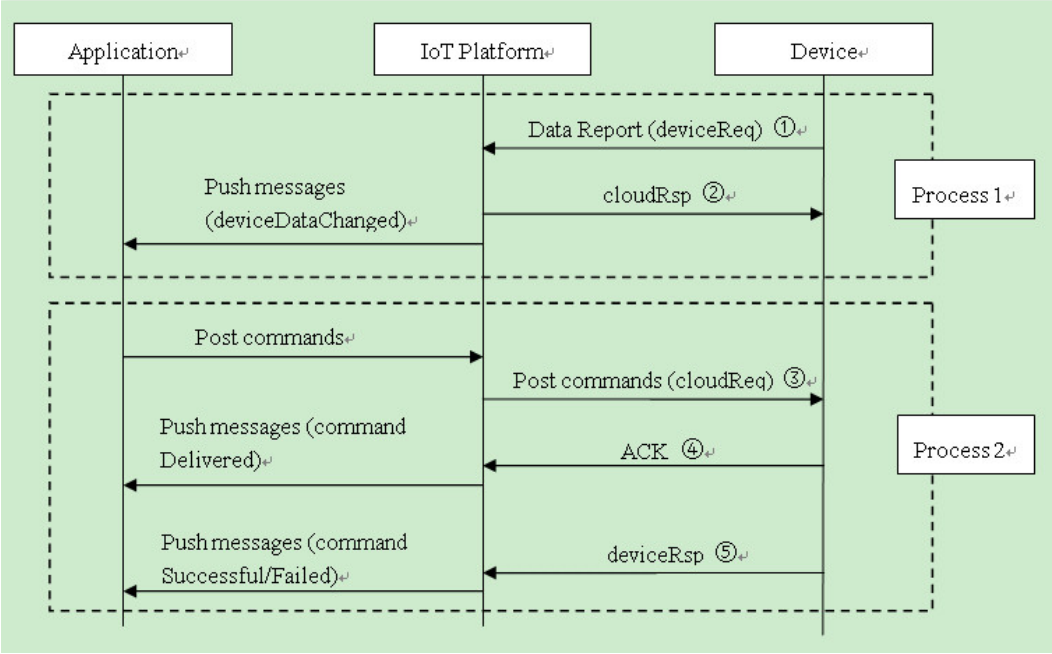


表 1-7 平台下发命令 encode 接口的入参结构定义

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识，IoT平台通过decode接口解析码流时获取该参数，通过encode接口编码时将该参数放入码流。	否
msgType	String	固定值"cloudReq"，表示平台下发的请求。	是
serviceId	String	服务的id。	是
cmd	String	服务的命令名，参见profile的服务命令定义。	是
paras	ObjectNode	命令的参数，具体字段由profile定义。	是
hasMore	Int	表示平台是否还有后续命令下发，0表示没有，1表示有。 后续命令是指，平台还有待下发的消息，以hasMore字段告知设备不要休眠。hasMore字段仅在PSM模式下生效，且需要“下行消息指示”开启。	是
mid	Int	2字节无符号的命令id，由IoT平台内部分配（范围1-65535）。 IoT平台在通过encode接口下发命令时，把IoT平台分配的mid放入码流，和命令一起下发给设备；设备在上报命令执行结果（deviceRsp）时，再将此mid返回IoT平台。否则IoT平台无法将下发命令和命令执行结果（deviceRsp）进行关联，也就无法根据命令执行结果（deviceRsp）更新命令下发的状态（成功或失败）。	是

示例：

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

表 1-8 平台收到设备的上报数据后对设备的应答 encode 接口的入参结构定义

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识，IoT平台通过decode接口解析码流时获取该参数，通过encode接口编码时将该参数放入码流。	否
msgType	String	固定值"cloudRsp"，表示平台收到设备的数据后对设备的应答。	是
request	byte[]	设备上报的数据。	是
errcode	int	请求处理的结果码，IoT平台根据该参数判断命令下发的状态。 0表示成功，1表示失败。	是
hasMore	int	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

说明

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

示例：

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

1.4.4.4 getManufacturerId 接口说明

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

示例：

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

1.4.4.5 getModel 接口说明

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

示例：

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

1.4.4.6 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";
return node;
}

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

正确示例：直接使用入参编解码，编解码库不做业务处理。

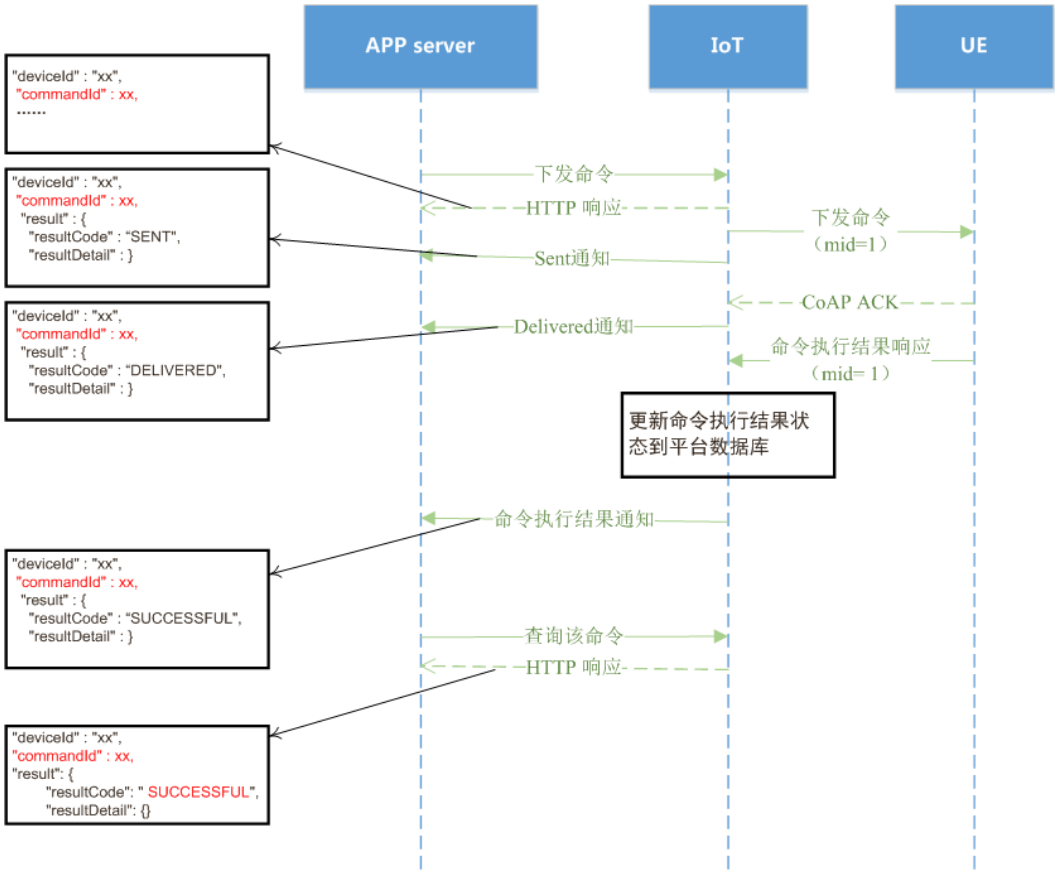
```
public class ProtocolAdapter {
@Override
public ObjectNode decode(finalbyte[] binaryData) throws Exception {
ObjectNode node;
  ...;
return node;
}

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


mid 字段的解释

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

实现mid

图 1-23 实现 mid 的消息流程

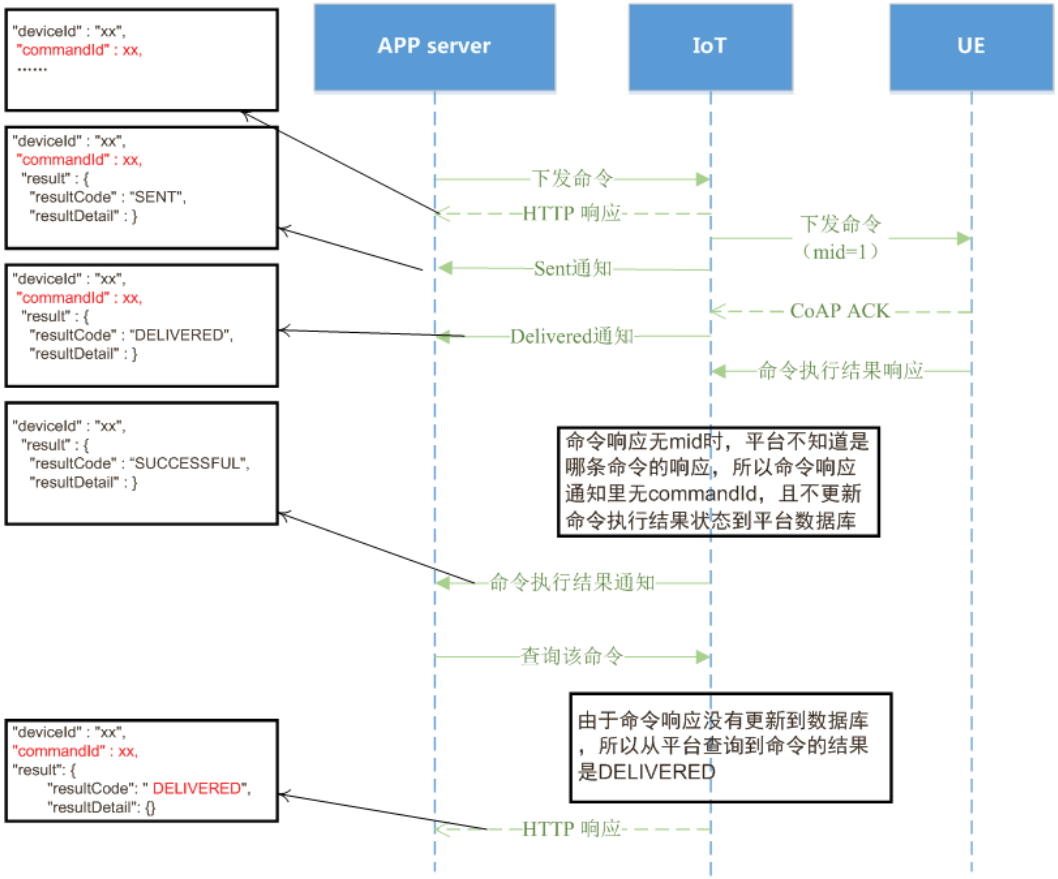


若实现了mid，并且命令执行结果已上报成功，则：

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

不实现mid

图 1-24 不实现 mid 的消息流程



若不实现mid，并且命令执行结果已上报成功，则：

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

说明

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

禁止使用 DirectMemory

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

错误示例：使用UNSAFE.allocateMemory申请直接内存

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

1.4.4.7 编解码插件的输入/输出格式

表 1-9 某款水表支持的服务定义

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

那么数据上报时decode接口的输出：

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

收到数据上报后，平台对设备的应答响应，调用encode接口编码，输入为

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

说明

request的取值[1,2]是模拟数据，以实际情况为准。

表 1-10 命令定义

基本功能名称	分类	名称	命令参数	数据类型	枚举值
WaterMeter	水表	-	-	-	-
-	CMD	SET_TEMPERATURE_READ_PERIOD	-	-	-
-	-	-	value	int	-
-	RSP	SET_TEMPERATURE_READ_PERIOD_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",
    "serviceId": "WaterMeter",
    "cmd": "SET_TEMPERATURE_READ_PERIOD",
    "paras": {
        "value": 4
    },
    "hasMore": 0
}
```

收到设备的命令应答后，调用decode接口解码，解码的输出

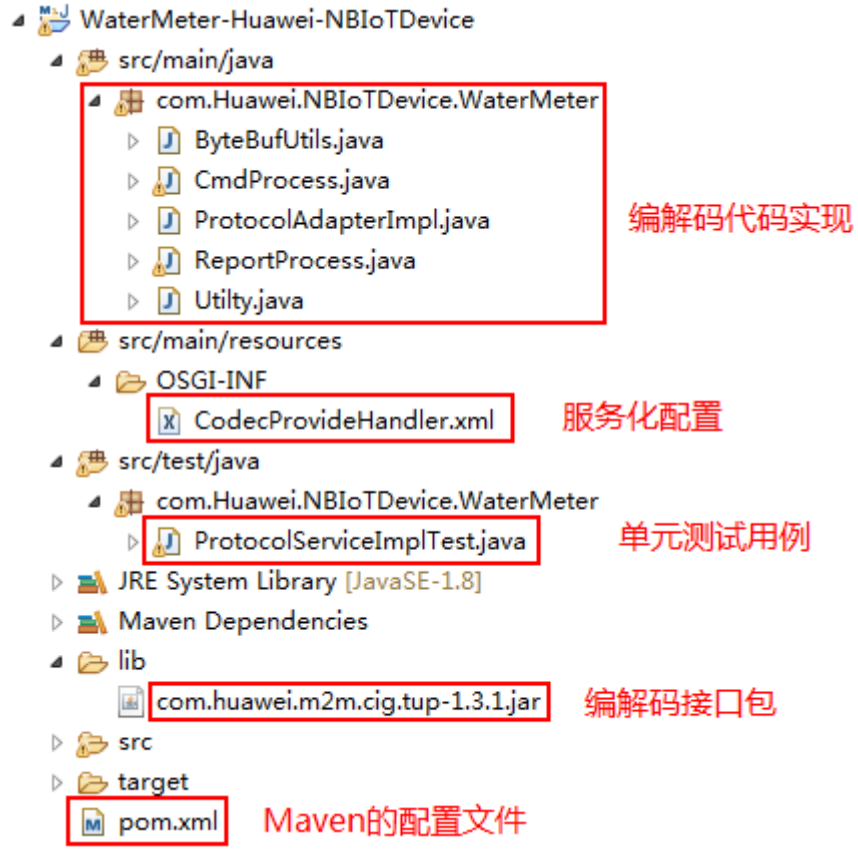
```
{
    "identifier": "123",
    "msgType": "deviceRsp",
    "errcode": 0,
```

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

1.4.4.8 实现样例讲解

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

图 1-25 样例工程结构图



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

说明

加密算法请使用JDK自带的加密算法，JDK支持的加密算法，详见**附录：JDK支持的加密算法**。

- Maven的配置文件

  在pom.xml文件中，根据命令规范，修改编解码插件的名字。

```
<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-NBIoTDevice -->
```

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

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<junit.version>4.11</junit.version>
<fasterxml.jackson.version>2.7.4</fasterxml.jackson.version>
<felix.maven.plugin.version>2.5.4.fixed2</felix.maven.plugin.version>
<json.lib.version>2.4</json.lib.version>
<m2m.cig.version>1.3.1</m2m.cig.version>
<slf4j.api.version>1.7.6</slf4j.api.version>
</properties>

<dependencies>
<!-- 单元测试使用 -->
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>${junit.version}</version>
</dependency>
<!-- 日志使用 -->
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>${slf4j.api.version}</version>
</dependency>
<!-- 转换JSON使用，必须 -->
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>${fasterxml.jackson.version}</version>
</dependency>
<!-- Huawei提供的编解码接口，必须 -->
<!-- systemPath请替换成你本地的目录 \codecDemo\lib\com.huawei.m2m.cig.tup-1.3.1.jar -->
<dependency>
<groupId>com.huawei</groupId>
<artifactId>protocal-jar</artifactId>
<version>1.3.1</version>
<scope>system</scope>
<systemPath>${basedir}/lib/com.huawei.m2m.cig.tup-1.3.1.jar</systemPath>
</dependency>

<!-- 本例中数据转换使用到的jar，你用到的jar请写到这里，记得把artifactId填入后面的Embed-
Dependency -->
<dependency>
<groupId>net.sf.json-lib</groupId>
<artifactId>json-lib</artifactId>
<version>2.4</version>
<classifier>jdk15</classifier>
</dependency>

</dependencies>
<build>
<plugins>
<!-- 编码需要使用JDK1.8版本 -->
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
<!-- OSGI规范打包配置 -->
<plugin>
<groupId>org.apache.felix</groupId>
<artifactId>maven-bundle-plugin</artifactId>
```



```
<version>${felix.maven.plugin.version}</version>
<extensions>true</extensions>
<configuration>
<buildDirectory>./target</buildDirectory>
<archive>
<addMavenDescriptor>false</addMavenDescriptor>
</archive>
<instructions>
<Bundle-RequiredExecutionEnvironment>J2SE-1.5</Bundle-RequiredExecutionEnvironment>
<Bundle-Activator></Bundle-Activator>
<Service-Component>OSGI-INF/*</Service-Component>
<!-- 请修改为你的编解码插件的名字，命名规范：设备类型-厂商ID-设备型号，例如：WaterMeter-Huawei-NBIoTDevice -->
<Bundle-SymbolicName>WaterMeter-Huawei-NBIoTDevice</Bundle-SymbolicName>
<Export-Package></Export-Package>
<!-- 把代码中import的package都引入进来，使用逗号分隔。【有两种jar不需要的填到Import-Package，否则插件无法启动，这两种jar分别为：①java.**开头的包；②引用的是Embed-Dependency里面的jar】
-->
<Import-Package>
org.slf4j,
org.slf4j.spi,
org.apache.log4j.spi,
com.huawei.m2m.cig.tup.modules.protocol_adapter,
com.fasterxml.jackson.databind,
com.fasterxml.jackson.databind.node
</Import-Package>
<!-- 除junit,slf4j-api,jackson-databind,protocal-jar， 其他所有的依赖包，必须把包对应的artifactId填入Embed-Dependency。artifactId之间以逗号分隔。Maven打包时你的依赖包需要打入到你的jar内。 -->
<Embed-Dependency>
json-lib
</Embed-Dependency>
</instructions>
</configuration>
<executions>
<execution>
<id>generate-resource</id>
<goals>
<goal>manifest</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

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

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

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

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

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

    public class CmdProcess {

        //private String identifier = "123";
        private String msgType = "deviceReq";
        private String serviceId = "Brightness";
    ```



```
private String cmd = "SET_DEVICE_LEVEL";
private int hasMore = 0;
private int errcode = 0;
private int mid = 0;
private JsonNode paras;


public CmdProcess() {
}

public CmdProcess(ObjectNode input) {

    try {
        // this.identifier = input.get("identifier").asText();
        this.msgType = input.get("msgType").asText();
        /*
        平台收到设备上报消息，编码ACK
        {
            "identifier":"0",
            "msgType":"cloudRsp",
            "request": ***,//设备上报的码流
            "errcode":0,
            "hasMore":0
        }
        * */
        if (msgType.equals("cloudRsp")) {
            //在此组装ACK的值
            this.errcode = input.get("errcode").asInt();
            this.hasMore = input.get("hasMore").asInt();
        } else {
        /*
        平台下发命令到设备，输入
        {
            "identifier":0,
            "msgType":"cloudReq",
            "serviceId":"WaterMeter",
            "cmd":"SET_DEVICE_LEVEL",
            "paras":{"value":"20"},
            "hasMore":0

        }
        * */
            //此处需要考虑兼容性，如果没有传mId，则不对其进行编码
            if (input.get("mid") != null) {
                this.mid = input.get("mid").intValue();
            }
            this.cmd = input.get("cmd").asText();
            this.paras = input.get("paras");
            this.hasMore = input.get("hasMore").asInt();
        }

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

}

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

```
                    buf.writeByte((byte) 0x72);
                    buf.writeByte((byte) brightlevel);

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

                    return byteRead;
                }
            }

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

}
```

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

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

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

public class ReportProcess {
    //private String identifier;

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

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

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

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

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



```
    private int mid = 0;

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

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

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

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

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



```
        }
    } else {
        return;
    }


}

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

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

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

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

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

            //serviceId=battery 数据组装
            ObjectNode batteryNode = mapper.createObjectNode();
            batteryNode.put("serviceId", "battery");
            ObjectNode batteryData = mapper.createObjectNode();
            batteryData.put("batteryVoltage", 25);
            batteryData.put("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;
        }
    }
}
```

1.4.4.9 附录：JDK 支持的加密算法

摘要算法

算法名称	算法	摘要长度	备注
MD	MD2	128	-
	MD5	128	-
SHA	SHA-1	160	-
	SHA-256	256	-
	SHA-384	384	-
	SHA-512	512	-
Hmac	HmacMD5	128	-
	HmacSHA1	160	-
	HmacSHA256	256	-
	HmacSHA384	384	-
	HmacSHA512	512	-

对称加密算法

算法名称	密钥长度	默认	工作模式	填充方式	备注
DES	56	56	ECB、CBC、PCBC、CTR、CTS、CFB、CFB8到128、OFB、OFB8到128	NoPadding、PKCS5Padding、ISO10126Padding	-

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

非对称加密算法

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

以及Base64。

1.5 开发应用

1.5.1 应用接入

概述

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

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

前提条件

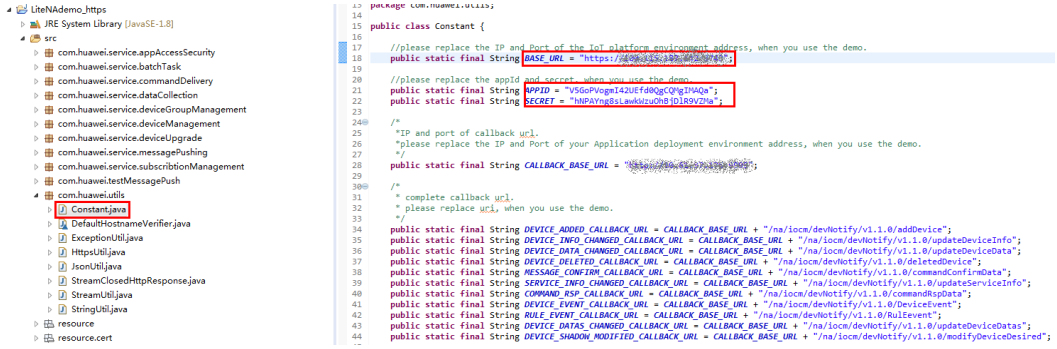
- 物联网平台已经部署相应的编解码插件。如果设备侧的“数据格式”为“JSON”，则不需要开发编解码插件。
- 应用侧通过HTTPS调用物联网平台的API前，需要集成相关证书，详见资源获取。
- 获取**调用API接口的代码样例（Java）**，并参考**准备Java开发环境**完成开发环境的配置和样例代码的导入。

操作指导

步骤1 参考**准备Java开发环境**章节，准备好Java开发环境。

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

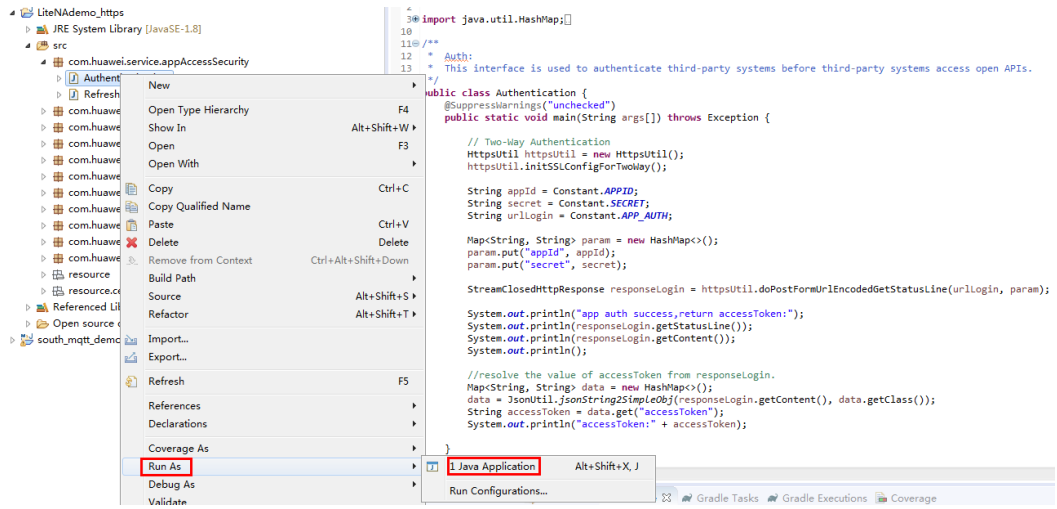
步骤2 在eclipse中，选择“src > com.huawei.utils > Constant.java”，修改BASE_URL、APPID、SECRET。



配置说明如下：

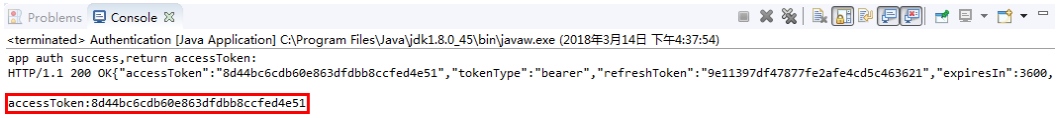
- BASE_URL：填写应用对接地址/端口号。
- APPID：填写创建应用或项目后获取的应用ID。
- SECRET：填写创建应用或项目后获取的应用密钥。

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



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

accessToken请妥善保存，以便于在调用其它接口时使用。



说明

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

----结束

1.5.2 订阅数据

概述

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

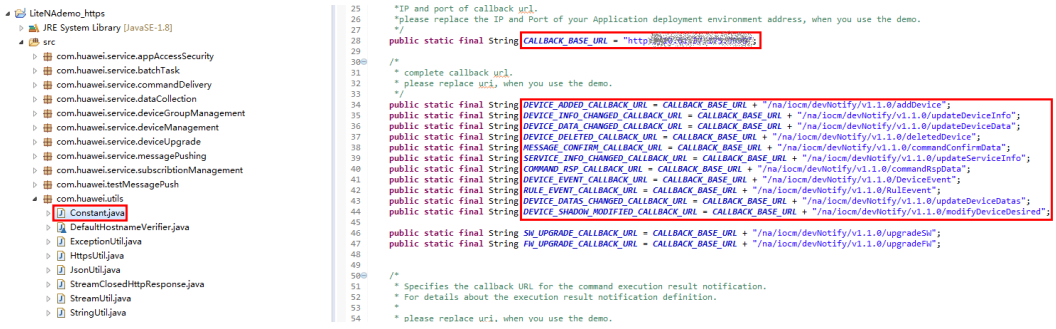
在订阅场景下，物联网平台是客户端，应用服务器是服务端，物联网平台调用应用服务器的接口，并向应用服务器推送消息。此时，如果订阅的回调地址为HTTPS地址，则需要在物联网上传CA证书。CA证书由应用服务器侧提供（证书获取方法可参考**导出CA证书**），并在开发中心的“应用 > 对接信息 > 应用安全 > 推送证书”上传，详见**上传CA证书**。

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

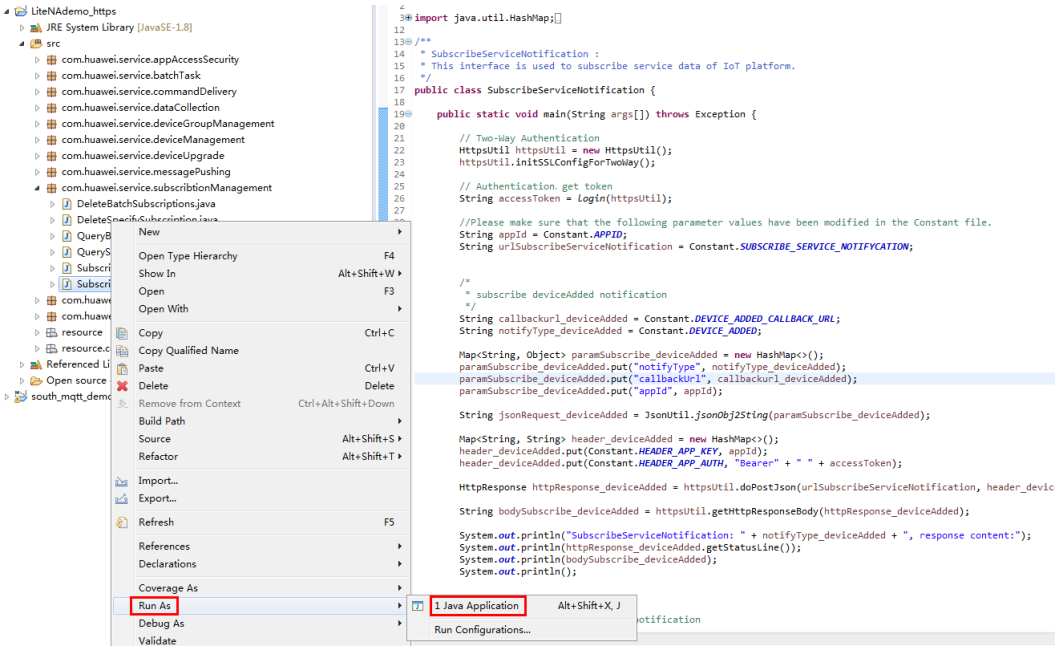
操作指导

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

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



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”。



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":"c5166051d466226b1f1a3590d17e4a3a","tokenType":"bearer","refreshToken":"357e88ea50a45d523b7e5ff9165cf58","expiresIn":3600,"

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

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

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

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

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

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

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

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

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

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

说明

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

----结束

1.5.3 注册设备

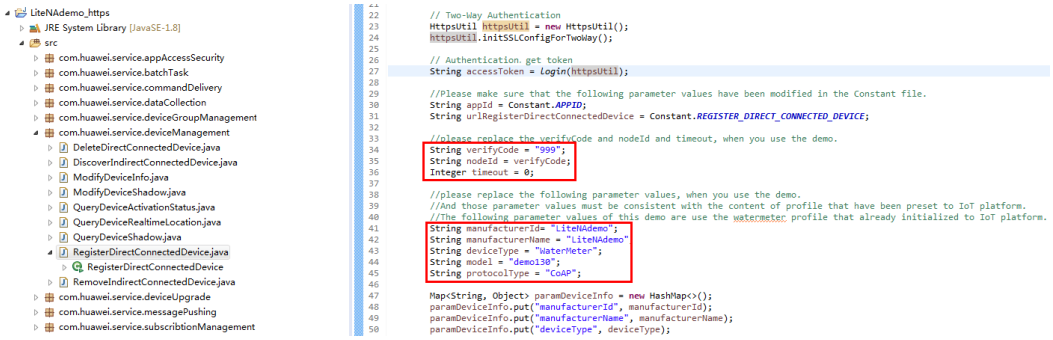
概述

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

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

操作指导

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



配置说明如下：

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

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

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

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

```
app auth success,return accessToken:
HTTP/1.1 200 OK{"accessToken":"1f337bfa6cb85f83243b99fda22d21b","tokenType":"bearer","refreshToken":"be3ccdce5390ed14b81c85f58e2712b2","expiresIn":3600,

RegisterDirectlyConnectedDevice, response content:
HTTP/1.1 200 OK{"deviceId":"d0ac5cac-d3af-4e0c-8166-5a67e1c6f0a7","verifyCode":"9999","timeout":0,"psk":"5a7f074ffdb1c46ed104a8efcafd0a0e"}
```

----结束

1.5.4 设备接入

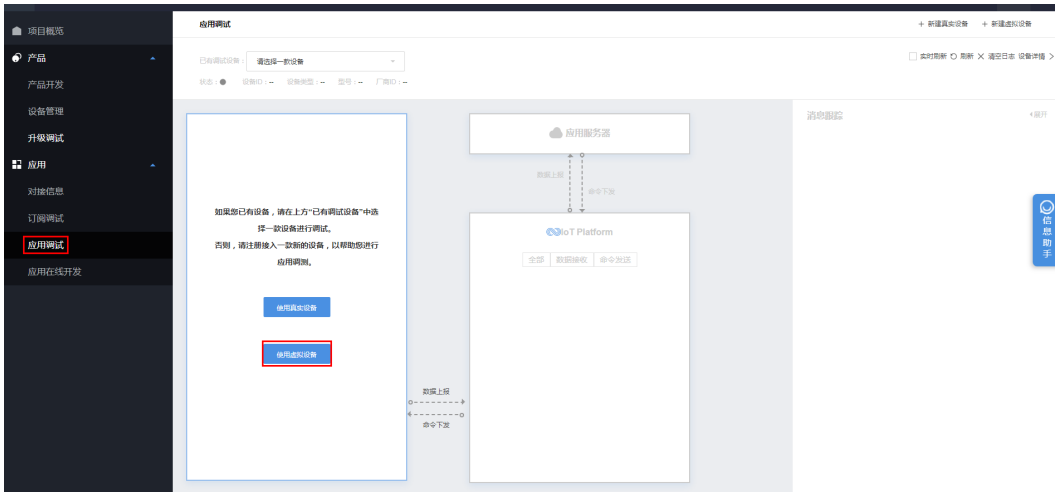
概述

设备接入物联网平台后，才可以经由物联网平台与应用服务器进行数据交互。

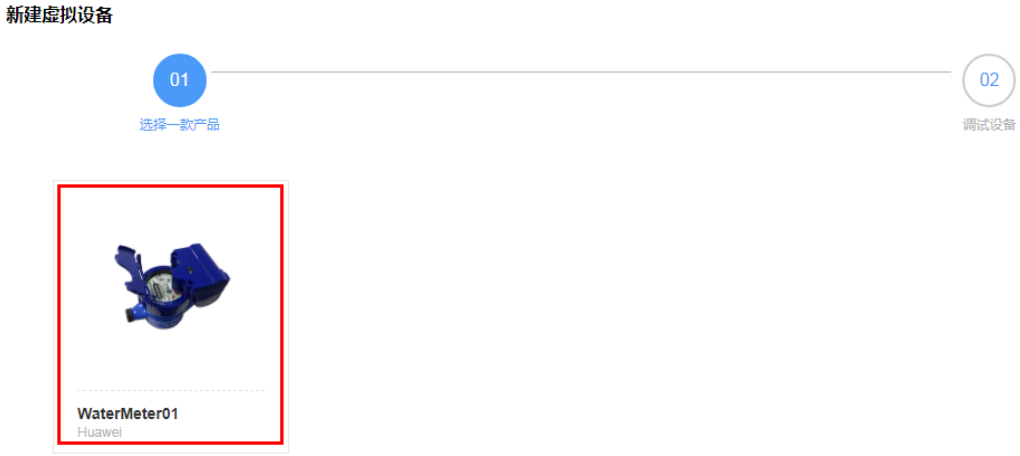
开发中心提供了应用调试功能，可以模拟设备接入物联网平台的场景。如果不使用开发中心的应用调试功能，请使用真实设备接入物联网平台。如下为使用开发中心模拟设备接入平台的操作指导：

操作指导

步骤1 选择“应用 ＞应用调试”，点击“使用虚拟设备”。



步骤2 系统将弹出“新建虚拟设备”窗口，根据提示选择一款产品。



----结束

1.5.5 数据上报

概述

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

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

操作指导

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

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

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

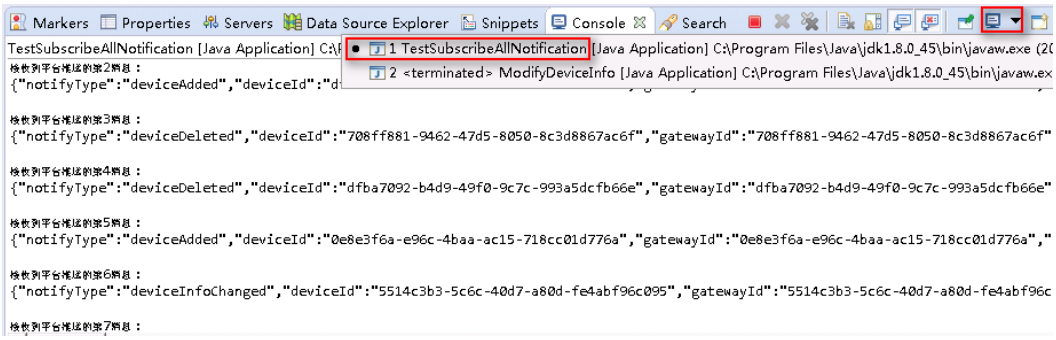
说明

如果不使用开发中心的应用调试功能，请使用真实设备进行数据上报。

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

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

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



----结束

1.5.6 命令下发

概述

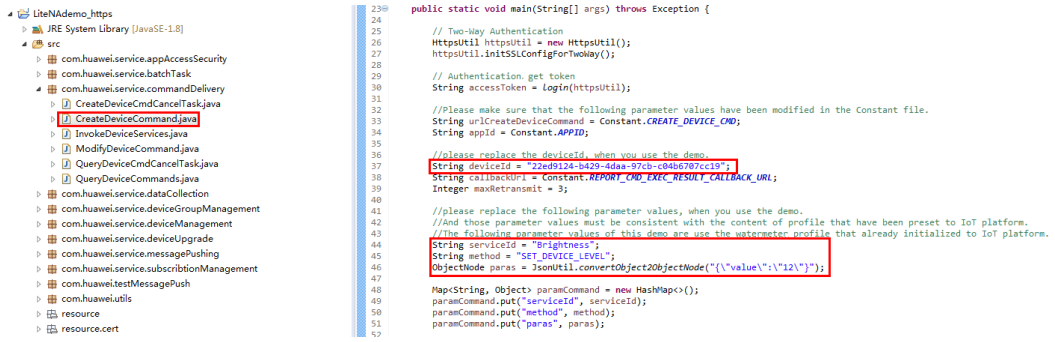
应用服务器需要调用物联网平台的“创建设备命令”接口或“设备服务调用”接口，对设备下发控制指令，接口信息详见API参考文档。

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

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

操作指导

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



配置说明如下：

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

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

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

```
app auth success,return accessToken:
HTTP/1.1 200 OK{"accessToken":"49b9b60d439d9f018a23c2c25d9e71","tokenType":"bearer","refreshToken":"51762486bb4df2142f904696a8ffb743","expiresIn":3600,"scope

PostAsynCommand, response content:
HTTP/1.1 201 Created{"commandId":"7e7ed2fd17c2449582c71adb0efa18b5","appId":"pQJNoCHou8mBo7anA7Fkw07luOAa","deviceId":"85ef387c-49cd-455d-9f70-a1bf1a990054",
```

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

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

----结束

1.5.7 其他接口

请参考API参考文档完成其他接口业务的开发。

1.5.8 参考信息

1.5.8.1 准备 Java 开发环境

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

1.5.8.1.1 安装 JDK1.8

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

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

1.5.8.1.2 配置 Java 环境变量（Windows 操作系统）

步骤1 右键单击“计算机”，选择“属性”。

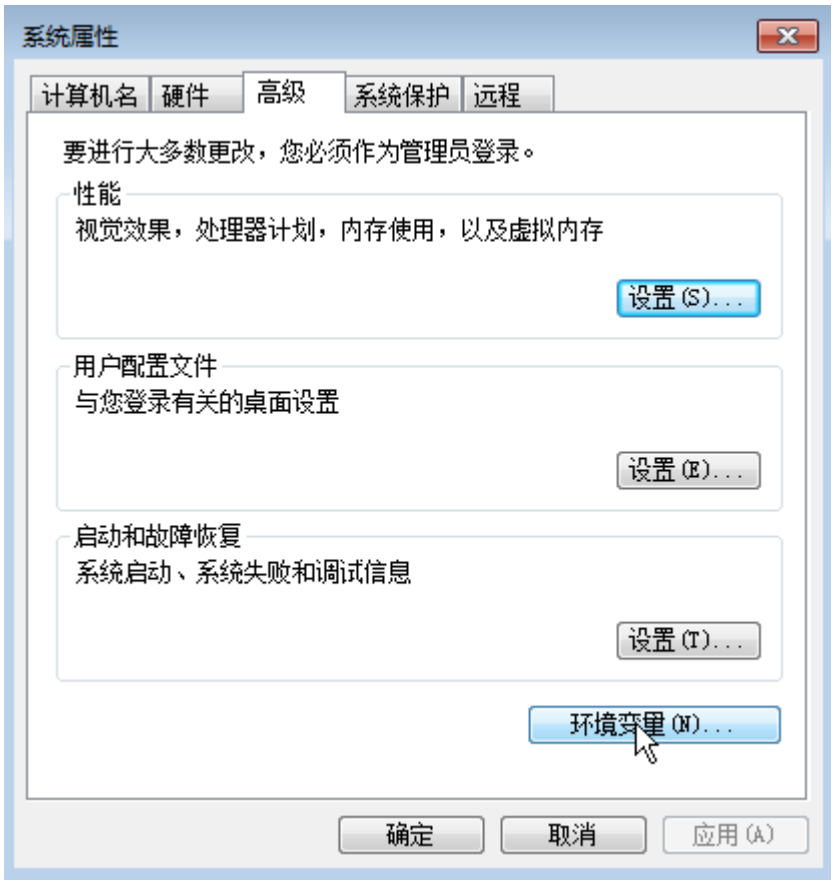
图 1-26 计算机属性

步骤2 点击“高级系统设置”。

图 1-27 系统

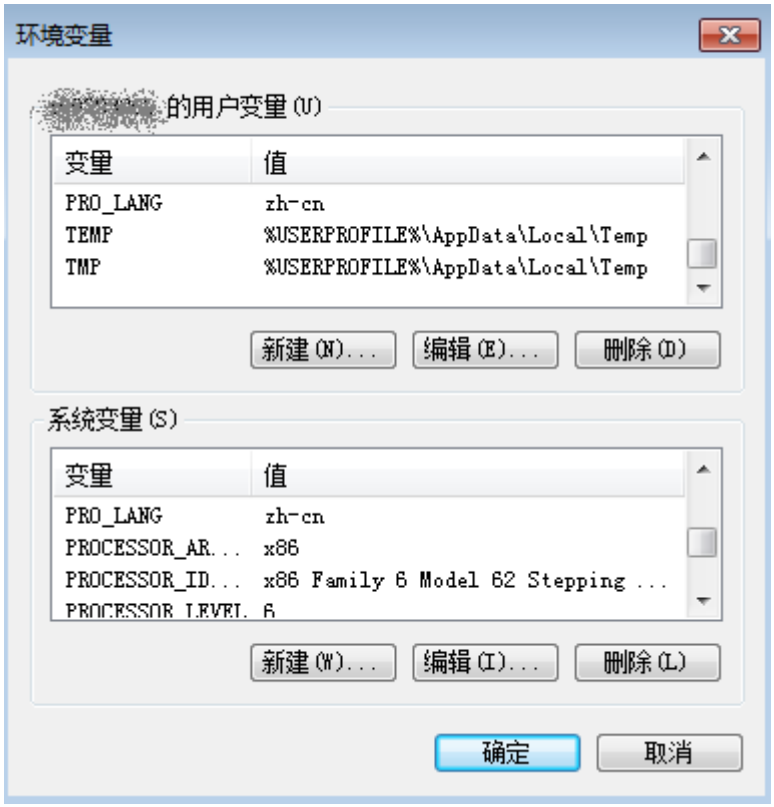
步骤3 点击“环境变量”。

图 1-28 系统属性



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

图 1-29 环境变量



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

图 1-30 新建 JAVA_HOME

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

两个路径之间需要使用“;”分割，分号是英文半角。

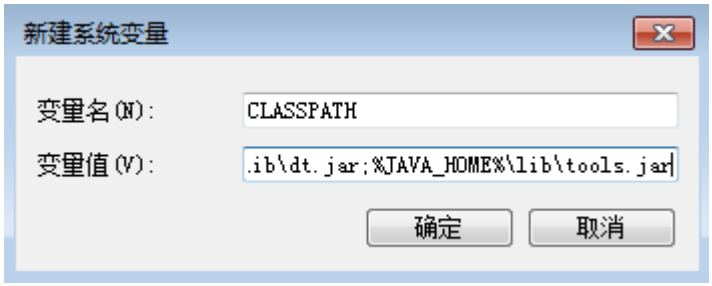
图 1-31 编辑 Path 变量

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

说明

路径以“.”开始，表示当前路径。

图 1-32 编辑 CLASSPATH 变量



步骤5 重启系统，使环境变量生效。

步骤6 选择“开始 > 运行”，输入“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。

图 1-33 验证环境变量

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

----结束

1.5.8.1.3 安装 Eclipse

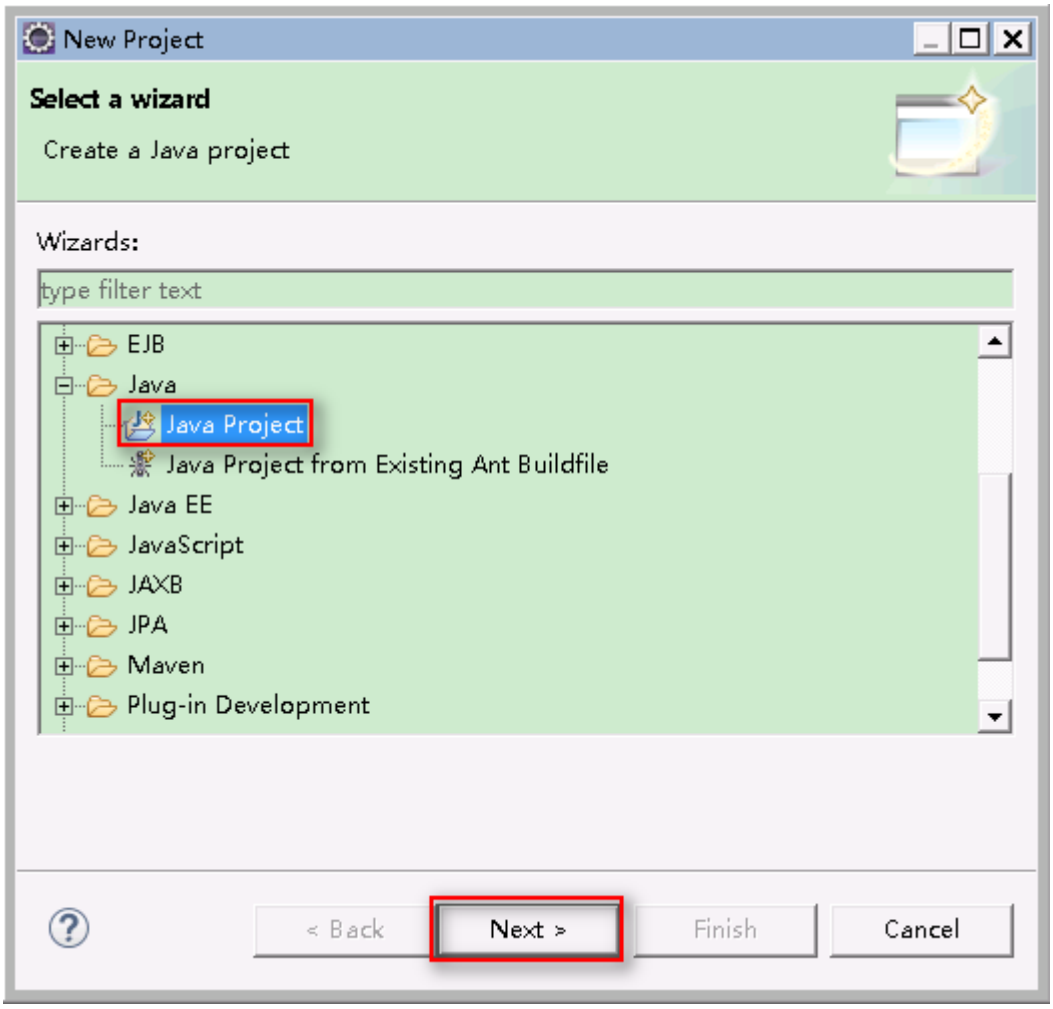
下载Eclipse安装包，直接解压缩到本地即可使用。

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

1.5.8.1.4 新建工程

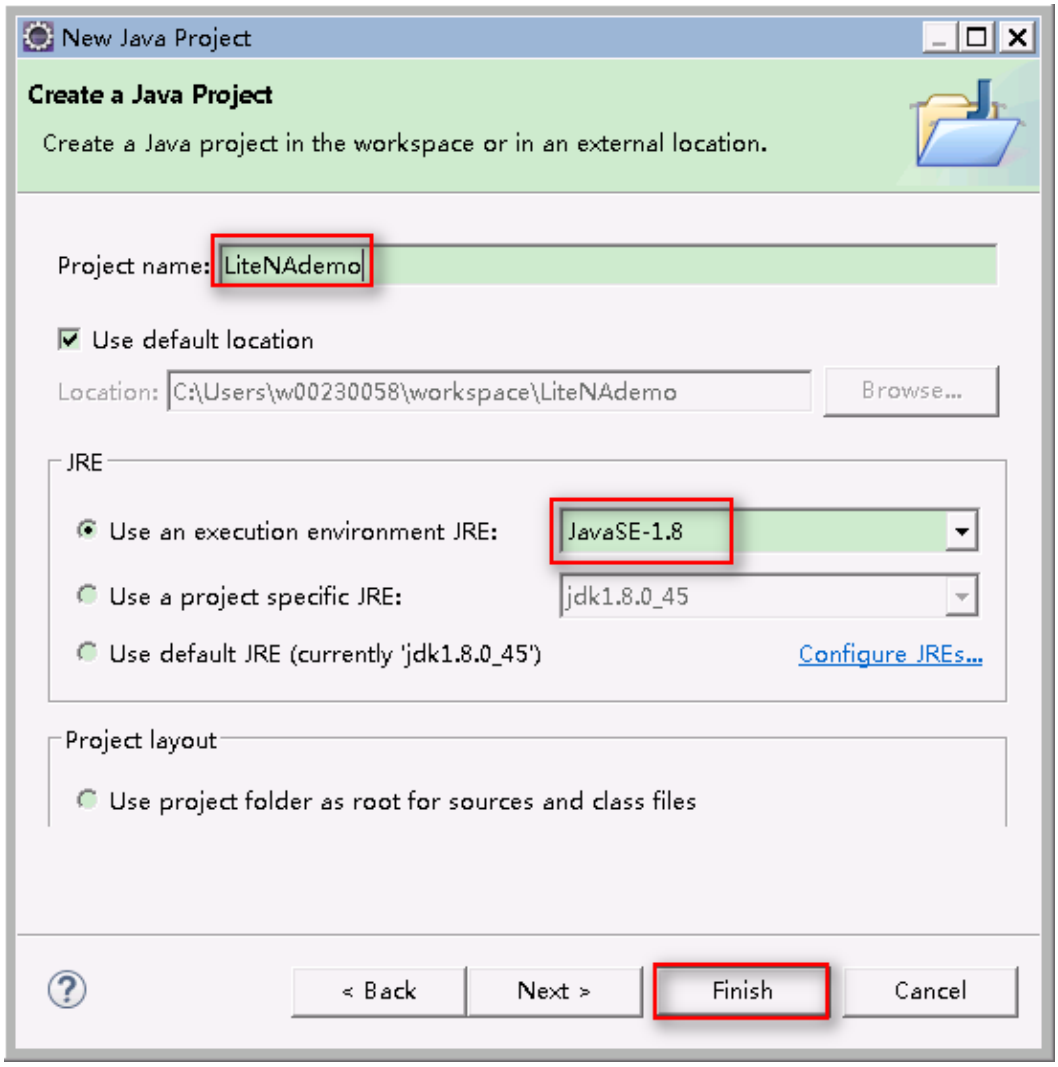
步骤1 运行Eclipse，选择“File > New > Project”，在弹出的对话框中选择“Java Project”，点击“Next”。

图 1-34 创建 Java 工程



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

图 1-35 配置工程名称



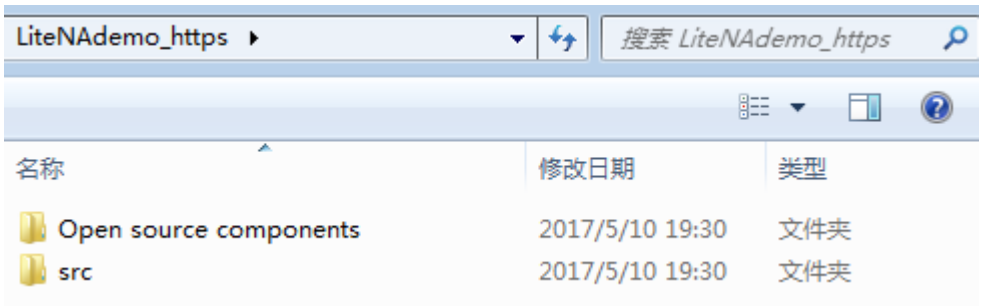
----结束

1.5.8.1.5 导入样例代码

步骤1 解压JAVA语言的接口调用样例。

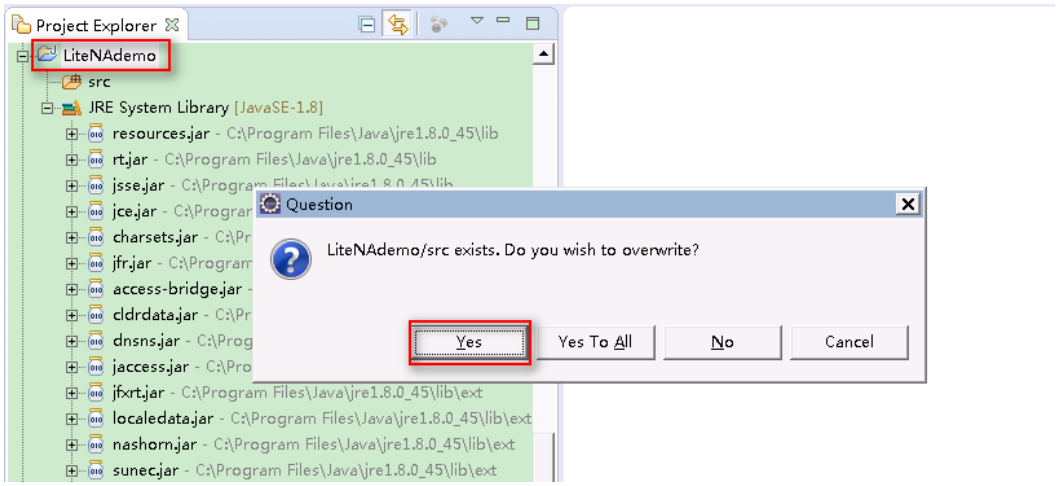
步骤2 完成解压后，拷贝（Ctrl+C）“Open source components”和“src”文件夹。

图 1-36 拷贝文件



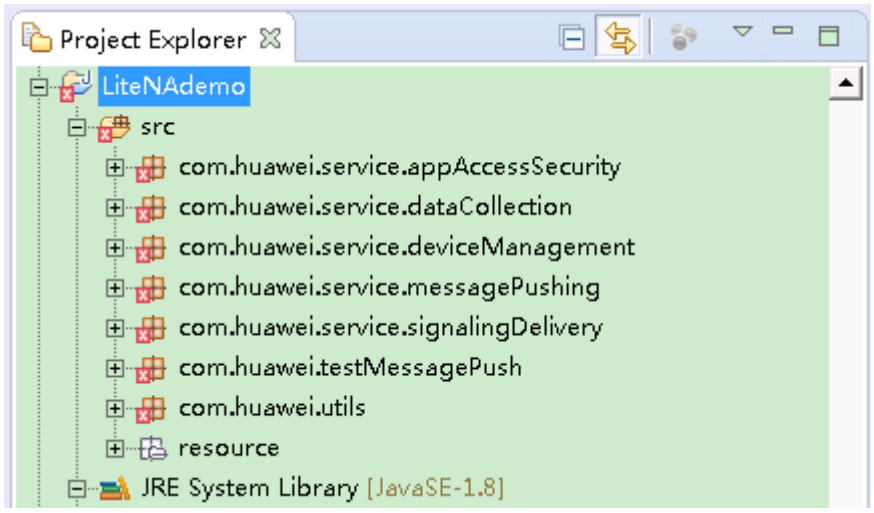
步骤3 打开在Eclipse创建的工程，点击选中工程名称，将拷贝的文件粘贴（Ctrl+V）到该工程目录下。

图 1-37 粘贴文件



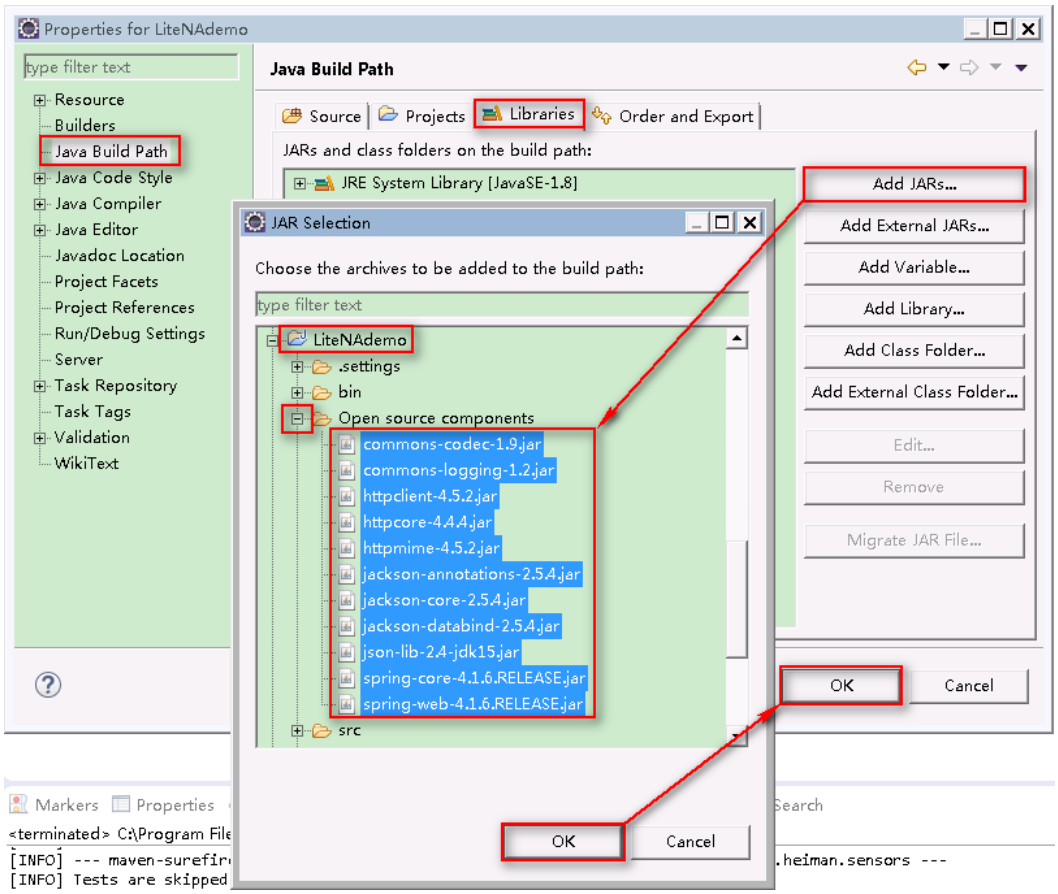
粘贴完成后，“src”目录下的文件存在错误。

图 1-38 “src”目录存在错误



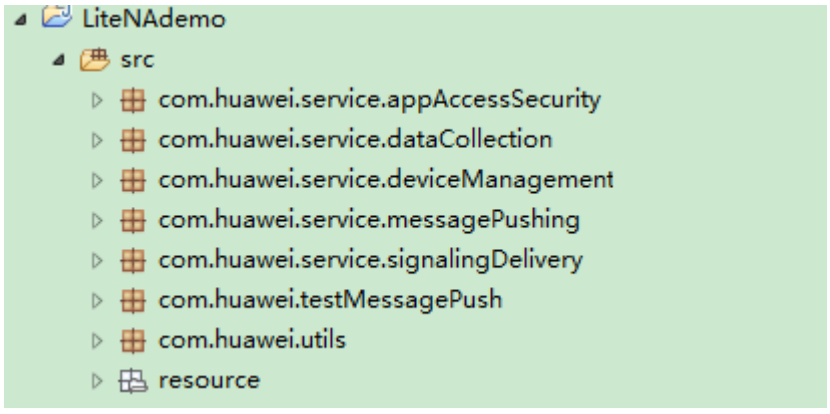
步骤4 右键单击工程名称，选择“Properties > Java Build Path > Libraries > Add JARs”，在弹出框中选中“Open source components”目录下的所有jar文件，点击“OK”。

图 1-39 导入 jar 文件



导入jar文件之后，“src”目录下文件的错误消失。

图 1-40 “src”目录错误消失



----结束

1.5.8.2 使用 Postman 测试平台北向接口

前提条件

在使用本方法前，您需要：

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

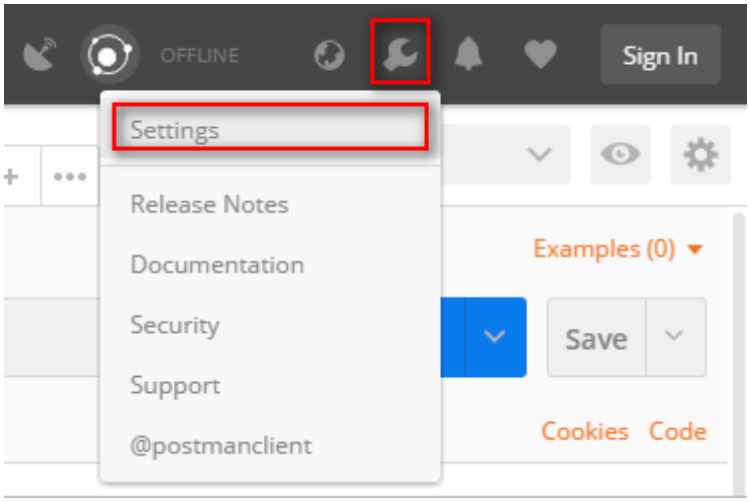
  说明

  Postman下载链接：**https://www.getpostman.com**

配置 Postman

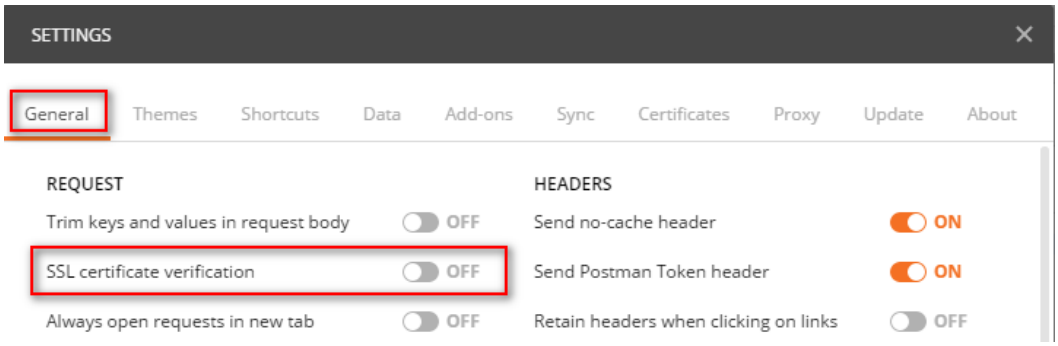
步骤1 打开Postman的“Settings”菜单。

图 1-41 进入“Settings”菜单



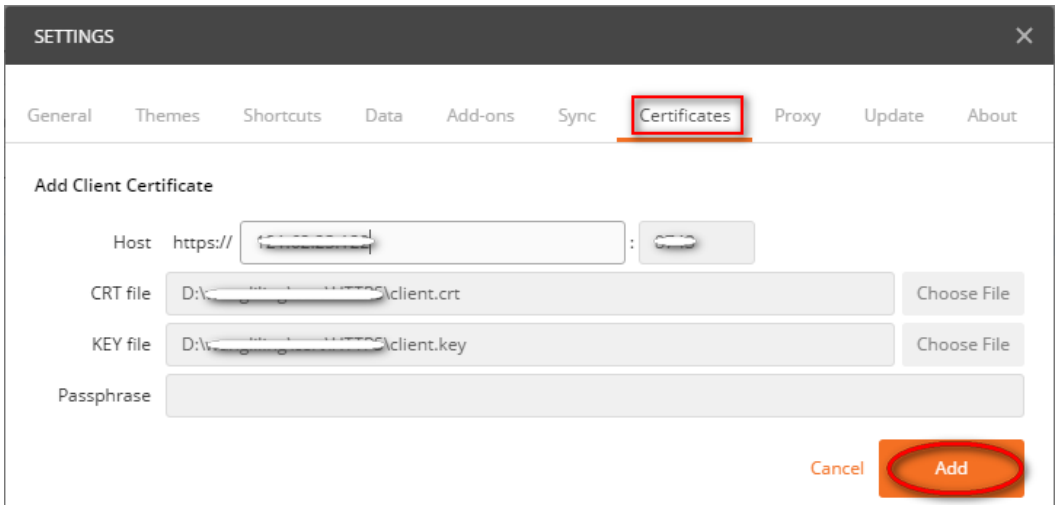
步骤2 关闭证书校验，使Postman不再校验服务端的证书。

图 1-42 关闭证书校验



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

图 1-43 配置客户端证书



----结束

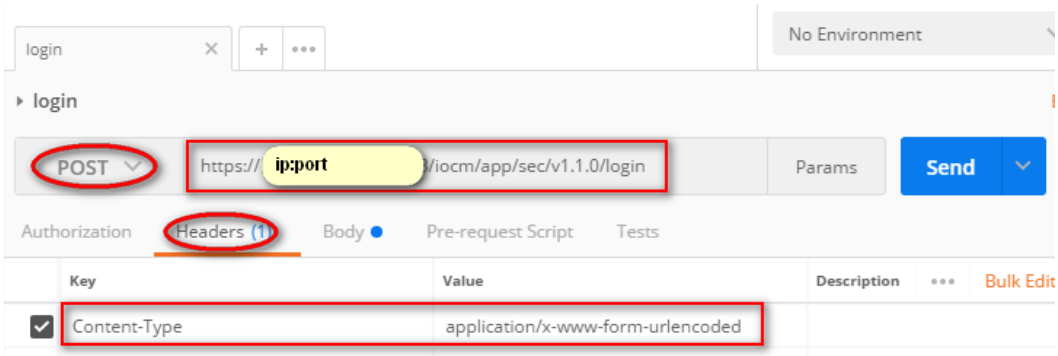
说明

client.crt与client.key为客户端证书和私钥文件。

调测“鉴权”接口

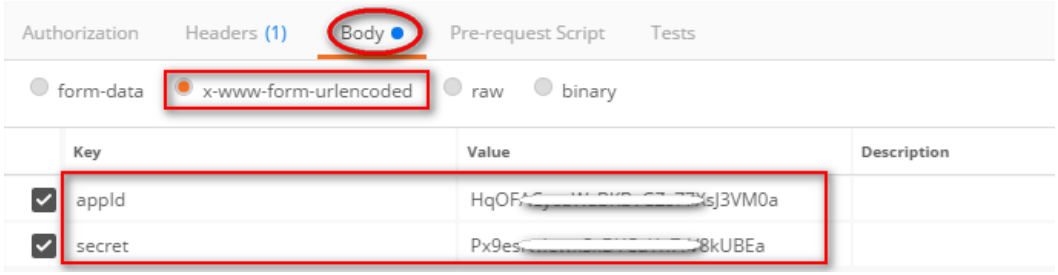
步骤1 配置“鉴权”接口的HTTP方法、URL和Headers。

图 1-44 配置 HTTP 方法、URL 和 Headers（鉴权）



步骤2 配置“鉴权”接口的Body。

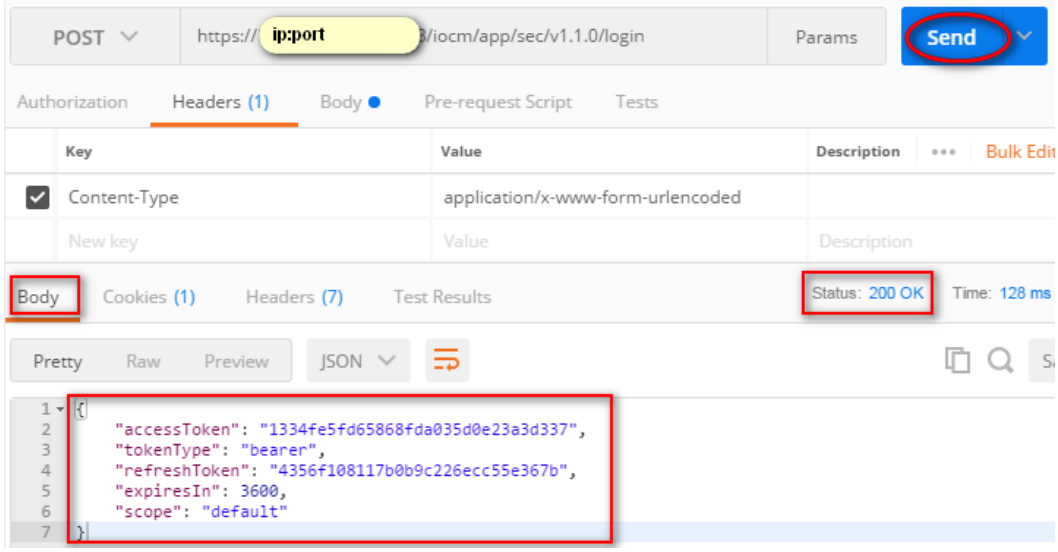
图 1-45 配置 Body（鉴权）



步骤3 点击“Send”，在下方查看返回码和响应消息内容。

请将返回的accessToken妥善保存，以便于在调用其它接口时使用。

图 1-46 查看响应信息（鉴权）

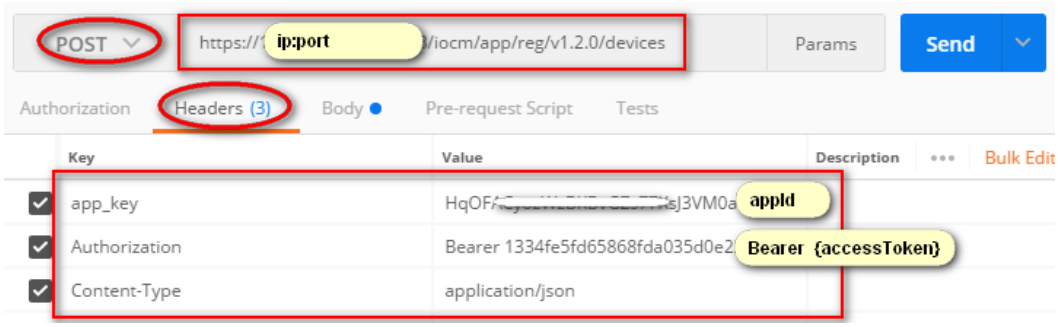


----结束

调测“注册直连设备”接口

步骤1 配置“注册直连设备”接口的HTTP方法、URL和Headers。

图 1-47 配置 HTTP 方法、URL 和 Headers（注册直连设备）



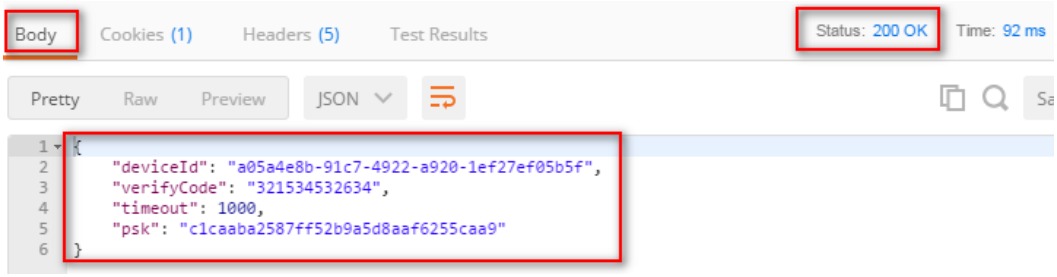
步骤2 配置“注册直连设备”接口的Body。

图 1-48 配置 Body（注册直连设备）

步骤3 点击“Send”，在下方查看返回码和响应消息内容。

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

图 1-49 查看响应信息（注册直连设备）



----结束

1.5.8.3 CA 证书

导出 CA 证书

应用服务器侧的CA证书，可以通过如下方式导出。

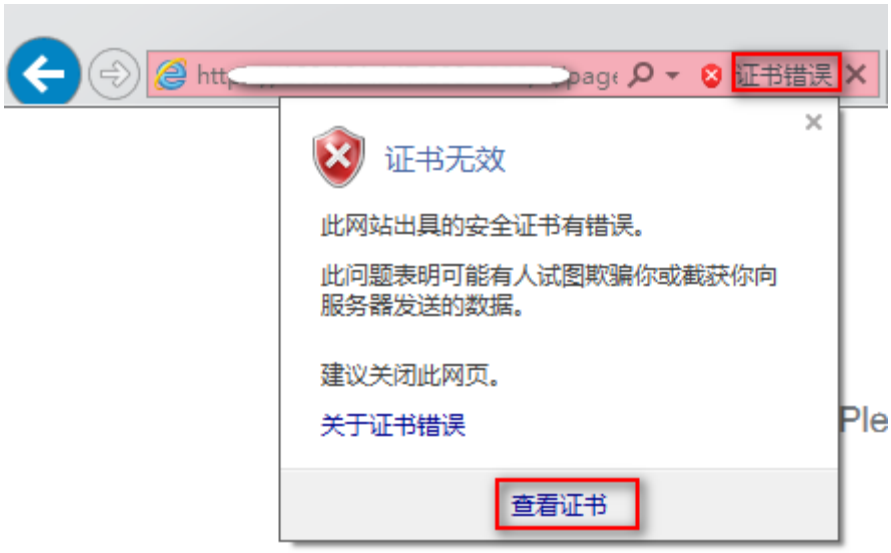
步骤1 使用浏览器打开回调地址，以IE为例。

步骤2 查看证书。自签名证书和非自签名证书的查看方式不同：

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

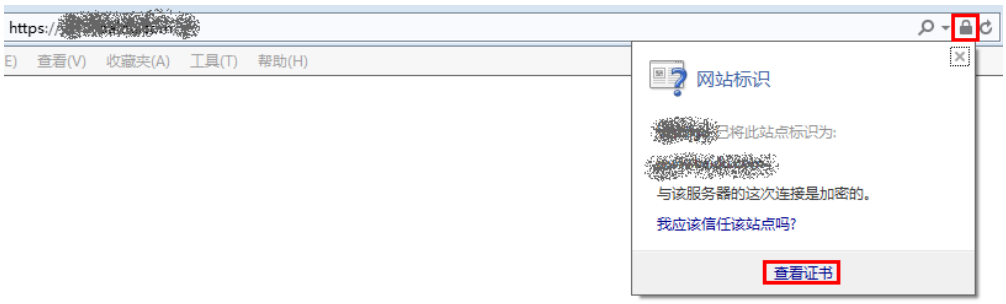
图 1-50 自签证书回调地址提示

图 1-51 查看自签证书



- 如果回调地址使用非自签证书，则选择“安全报告 > 查看证书”。

图 1-52 查看非自签证书



步骤3 在“证书路径”中查看证书级别，当前查看的为证书的最后一级。

图 1-53 证书路径

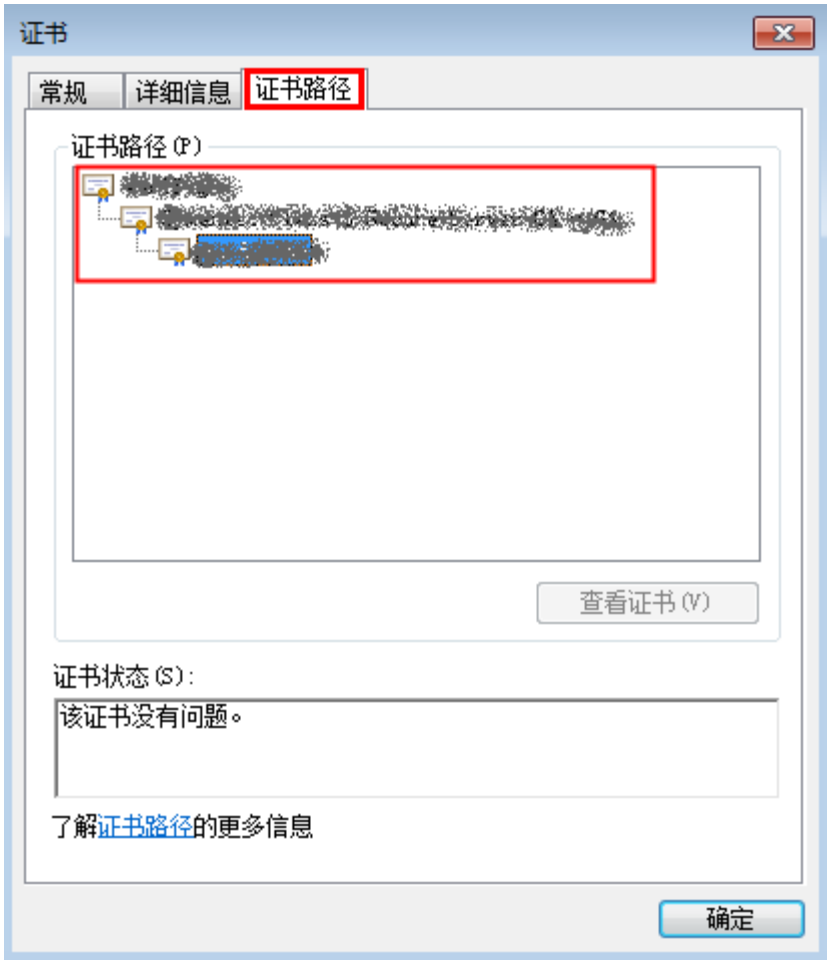
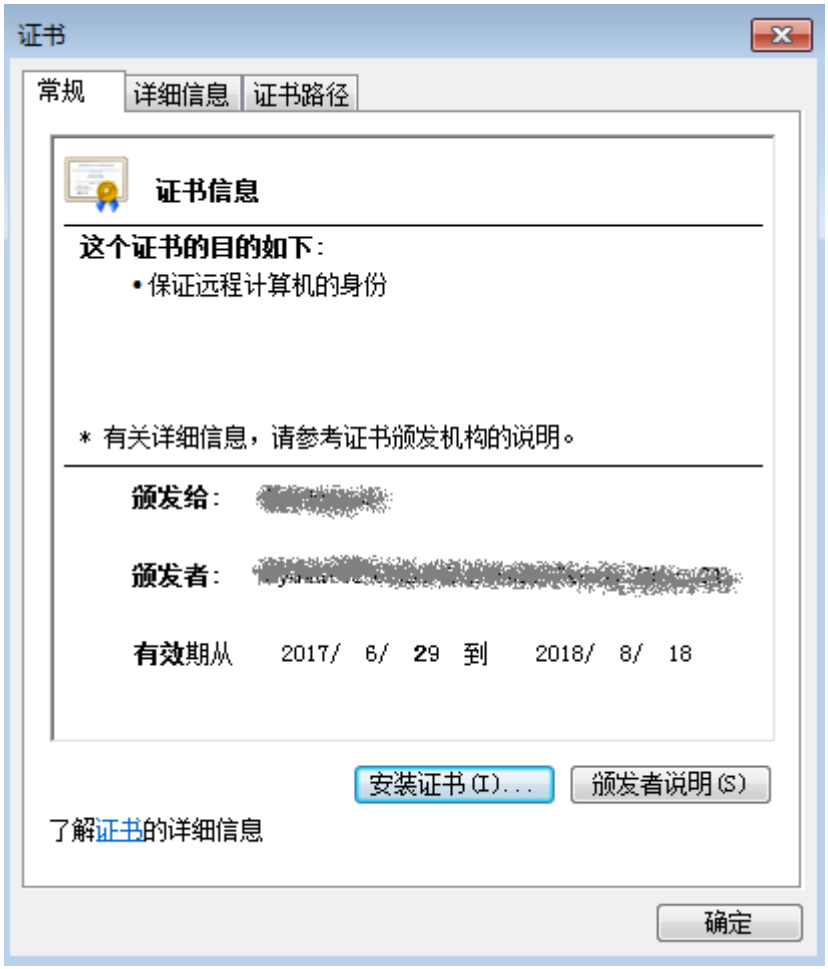


图 1-54 证书常规信息



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

图 1-55 证书详细信息

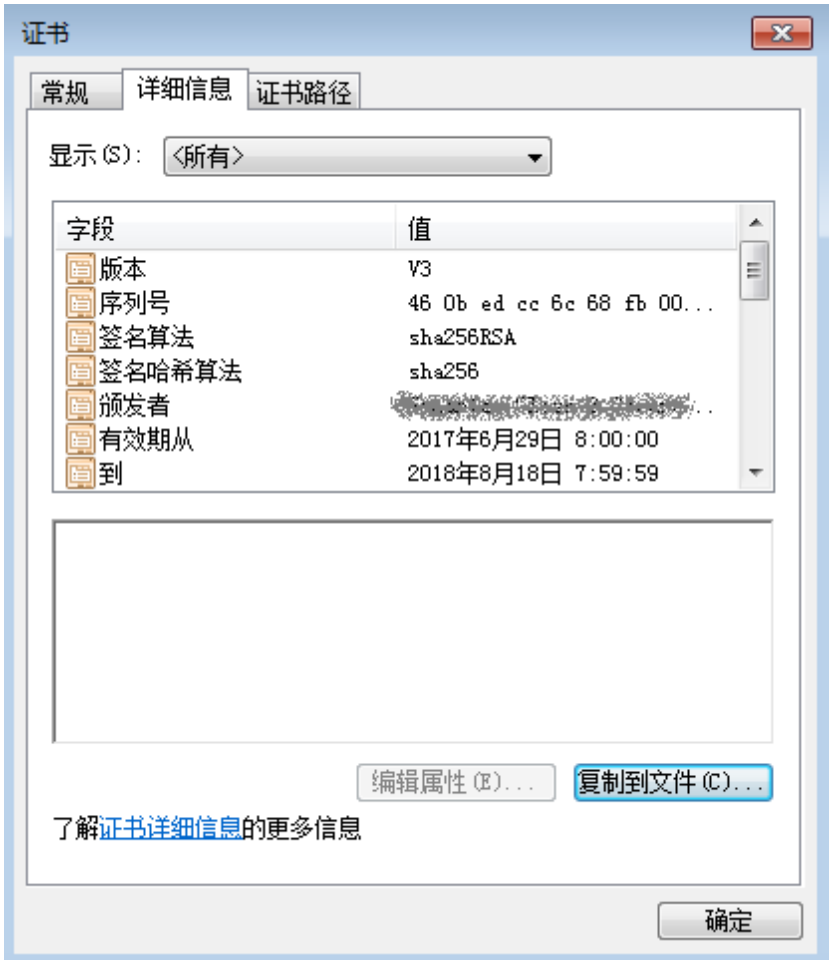
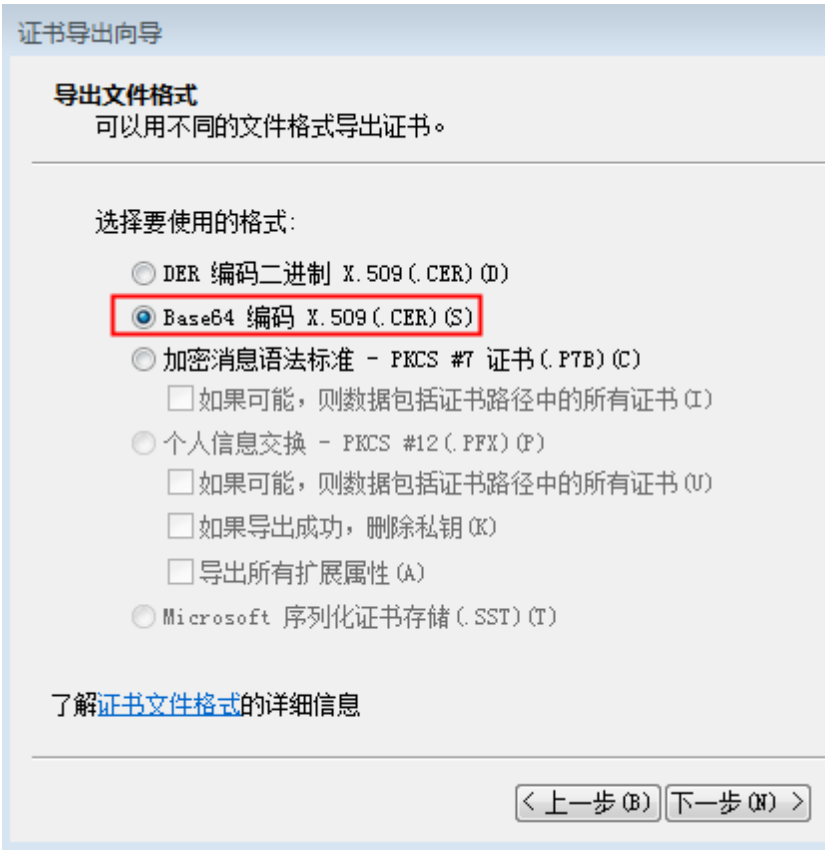


图 1-56 选择证书导出格式



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

步骤6 重复步骤5的操作，直到完成所有级别证书的导出。

步骤7 使用文本编辑器，将所有导出的证书以首尾相连的方式，合并到一个文件。

说明

合并的文件之间不能存在换行符。

图 1-57 合并证书

```
pLkJkLQYWBSHuxOizGdwCjw1mAT5G9+443fNDsgN3BAAAAFc8uXxRAAABAMARjBE
AiA+PkSvZOvIBmxaBLXLceHH1NqW+Mcrz+xtXNmJOl2ElQIgfVeG4xeuWrb7bpoU
smR+LStIG1TPCwimn3JRE3we/fYwDQYJKoZIhvcNAQELBQADggEBADjrCz8a7cax
h7vpyuUFZ/fiKBHE7VLqfppgf3XYNBoqh21qM6gTGzdiSeZj+vx+KOUn38f080Rg
N2aEkag3n03cufIXR8Yn8haXcusz5PONSlMQnN5rZBwpZ8obItiO8KGOh5lgHQ+s
SloX/j8nDDCQgrNkcG2A78nUT+VxGGENxnPmqajP/O2h/kg02qjcnPoj6Elmm/At
5dWWANX374yS7c0fgLZZ1mfZoIqooaRxsSJl5RzyRNU3Bzv5CZCJCGYFqC3RS28Q
vTCjde7TMsAQiWkZ97IKlUMXdbHManm7K85aWcG4Wg8isr9d2GPUZYgcUSc8KfWY
aP5MzoeU6ug=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
```

步骤8 将合并后的证书文件后缀修改为pem。

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

图 1-58 上传证书

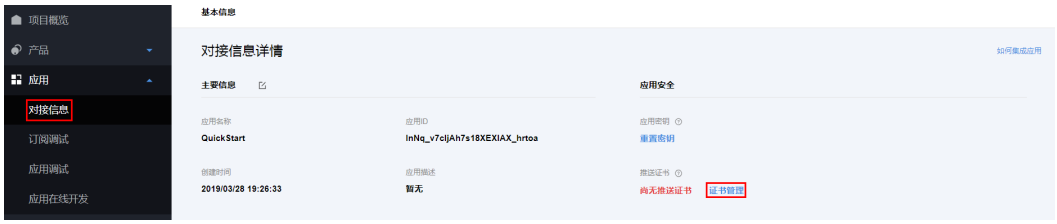
----结束

上传 CA 证书

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

在开发中心上传CA证书

步骤1 选择“应用 >对接信息”，在“推送证书”区域，点击“证书管理”。



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

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

----结束

在管理门户上传CA证书

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

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

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

----结束

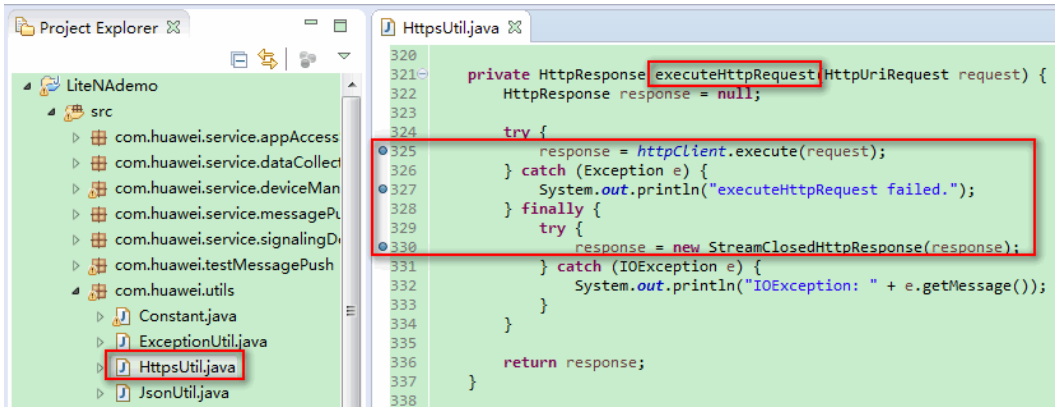
1.5.8.4 单步调测

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

步骤1 在最终发出http/https消息的代码处设置断点。

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

图 1-59 设置断点

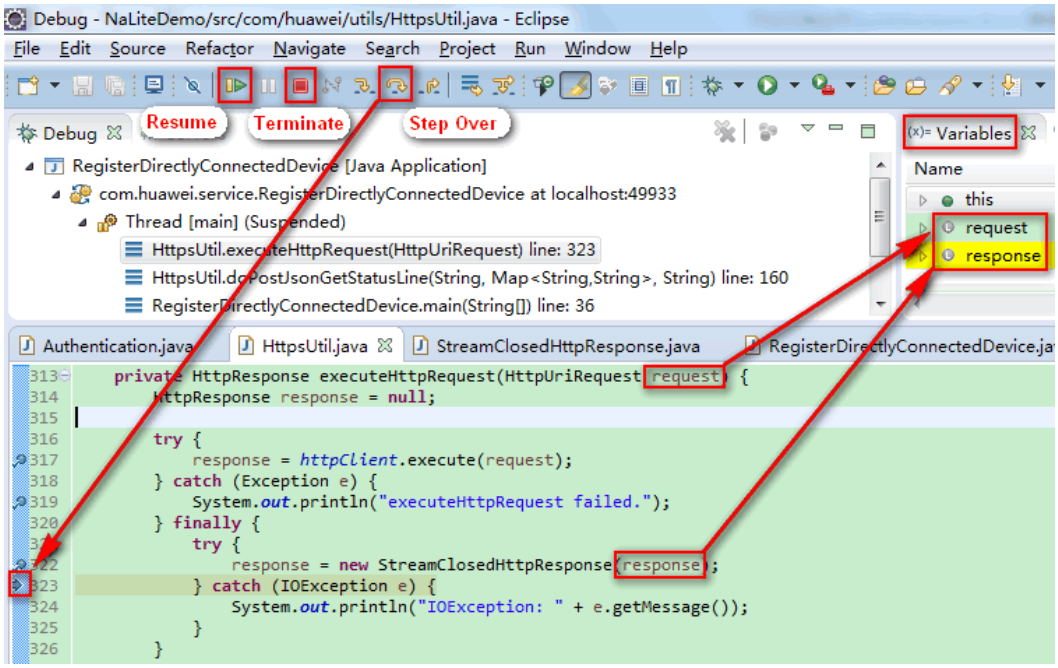


步骤2 右键单击需要调测的类，例如：“Authentication.java（根据您建立的工程类型进行选择）”，选择“Debug As > Java Application”。

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

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

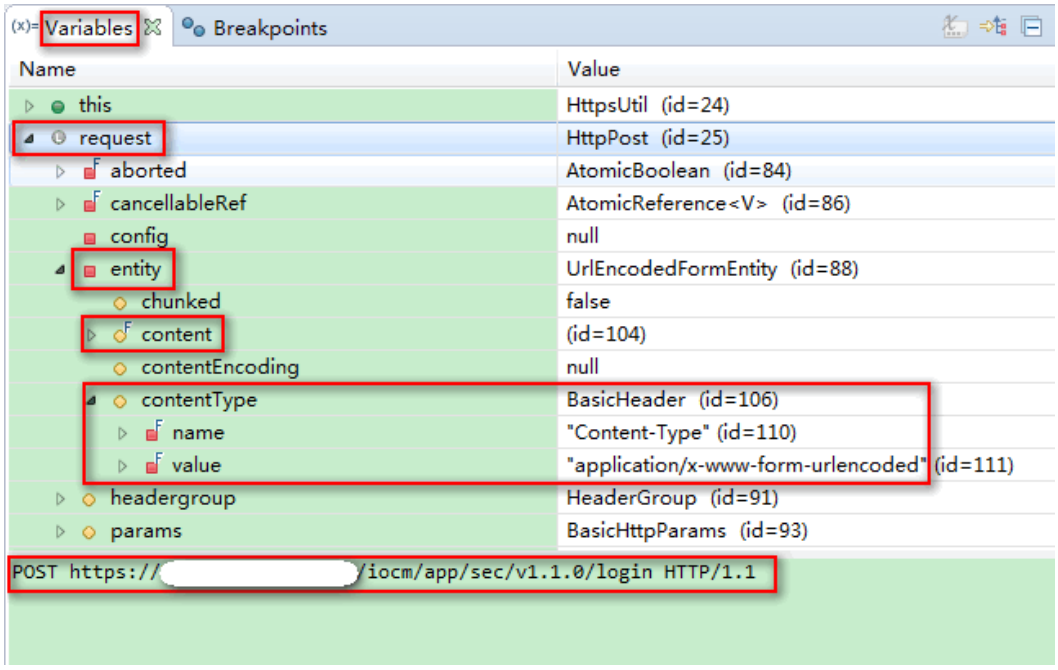
图 1-60 单步调试



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

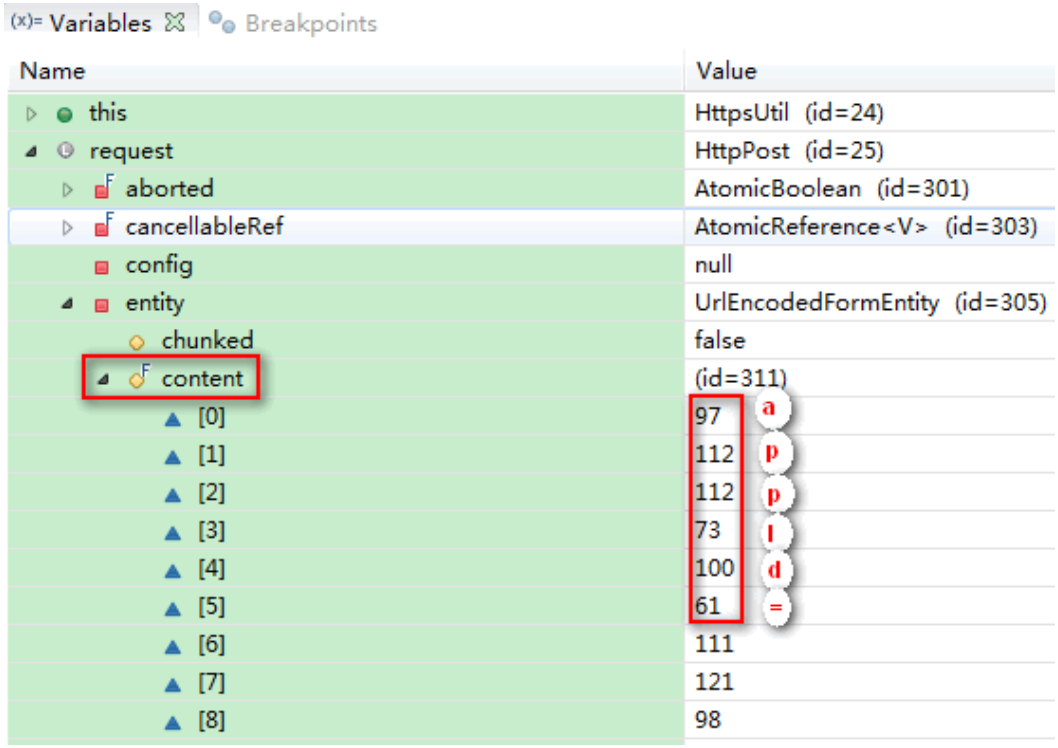
选中“request”变量时，可以在下方内容展示区看到应用程序发送请求的URL；在“entity”中可以看到发送的消息内容。

图 1-61 展开“request”变量



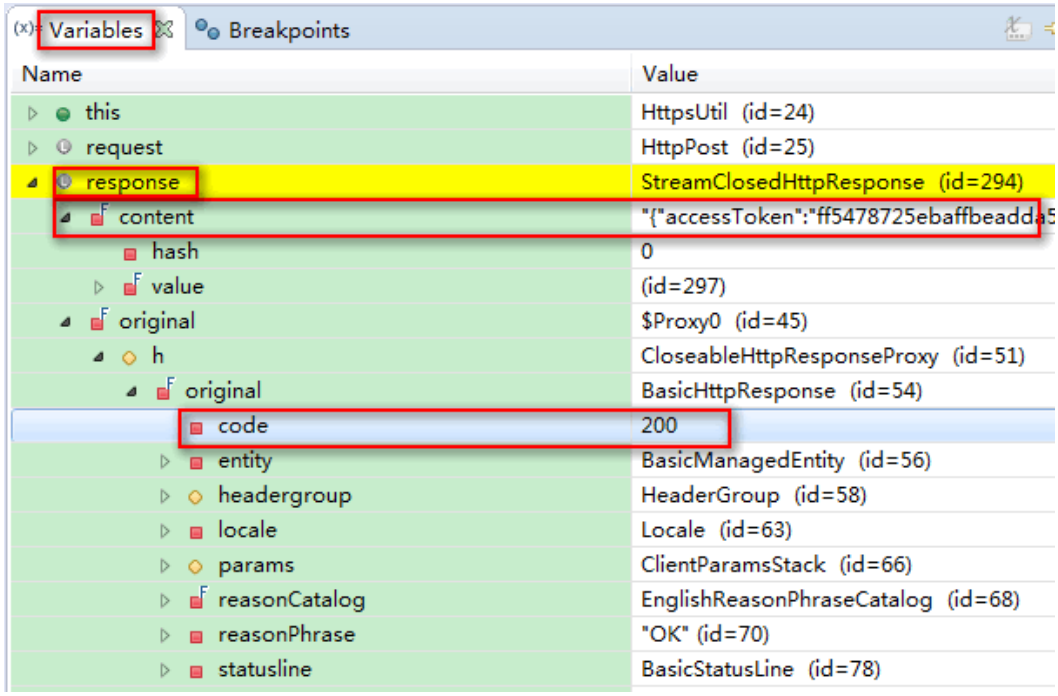
应用ID（appId）和应用密钥（secret）在“content”字段内，使用十进制的ASCII码表示，需要对照ASCII码表将其转化为字母和符号。

图 1-62 查看“content”字段



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

图 1-63 展开“response”变量



说明

在代码样例中，“Authentication.java”之外的类均会先调用鉴权接口。因此，在对“Authentication.java”之外的类进行单步调测时，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

1.6 开发设备

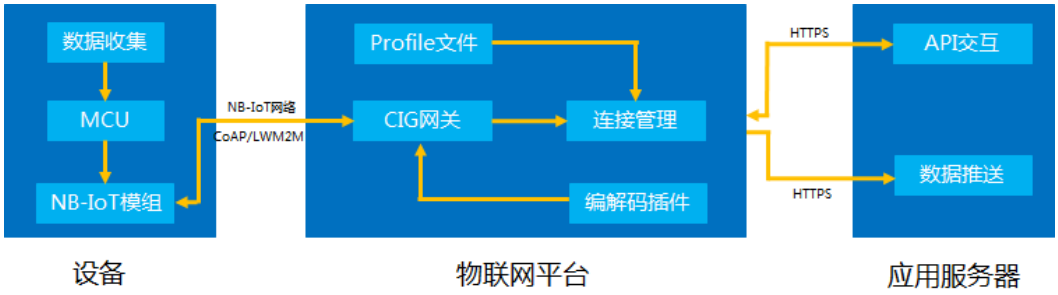
1.6.1 LWM2M/CoAP 设备集成

1.6.1.1 设备集成

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

集成 NB-IoT 模组

集成NB-IoT模组的设备，可以通过NB-IoT网络接入物联网平台。



特点	● 覆盖广，相比LTE提升20dB以上的增益 ● 低功耗，聚焦小数据量、小速率应用 ● 海量连接，单扇区支持5万个连接 ● 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
应用场景	对数据时效性要求低，数据包较小，设备位置变化较小，需要电池供电，例如：智能抄表，智能路灯等。
网络需求	● NB-IoT网络：由运营商构建。 ● NB-IoT SIM卡：向NB-IoT网络运营商购买。 ● NB-IoT模组：向模组厂商购买。
通信协议	CoAP/LWM2M
相关资源	请从模组厂商获取更多信息和支撑。

集成 LiteOS SDK

LiteOS SDK是用于设备侧集成的轻量化SDK，它的具体特征如下：

特点	● 屏蔽了协议和安全细节，用户可以专注自身的应用，无需关注协议和安全的具体实现。 ● 提供适配层，用户只需适配少量接口，便可以将LiteOS SDK进行移植。 ● 支持对终端设备上报的数据进行缓存，且具备重传和确认机制，保障数据上报的可靠性。 ● 支持固件升级，并实现了断点续传、固件包完整性保护。 ● 支持安全和非安全两种连接方式。
运行环境	RAM > 32KB FLASH > 128KB
网络需求	NB-IoT、2/3/4G、有线网络等。
通信协议	CoAP、LWM2M
相关资源	如果您选择在智能终端中集成LiteOS SDK，请参考**LiteOS SDK使用指南**。

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命令	作用	备注
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 +NCDP="IP","port"	设置设备对接的IoT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命令	作用	备注
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IoT平台(LWM2M SERVER)接收到的数据。其他芯片或模组厂商可参考实现。

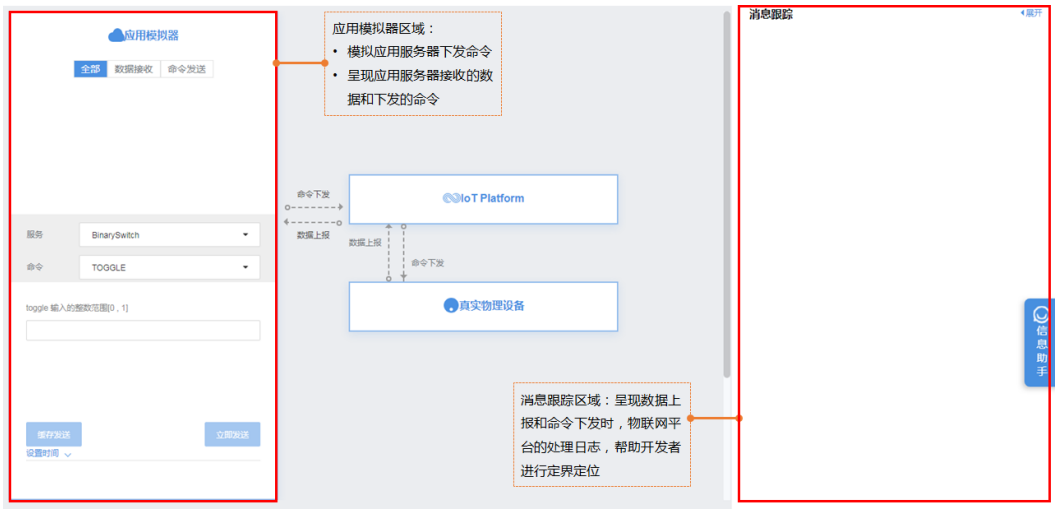
1.6.1.2 设备调测

概述

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

进行在线调测时，可以使用真实设备调测，也可以使用虚拟设备调测：

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



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

使用真实设备调测

步骤1 在产品开发空间，点击“在线调测”。

步骤2 在“设备列表”区域，点击“新增测试设备”。

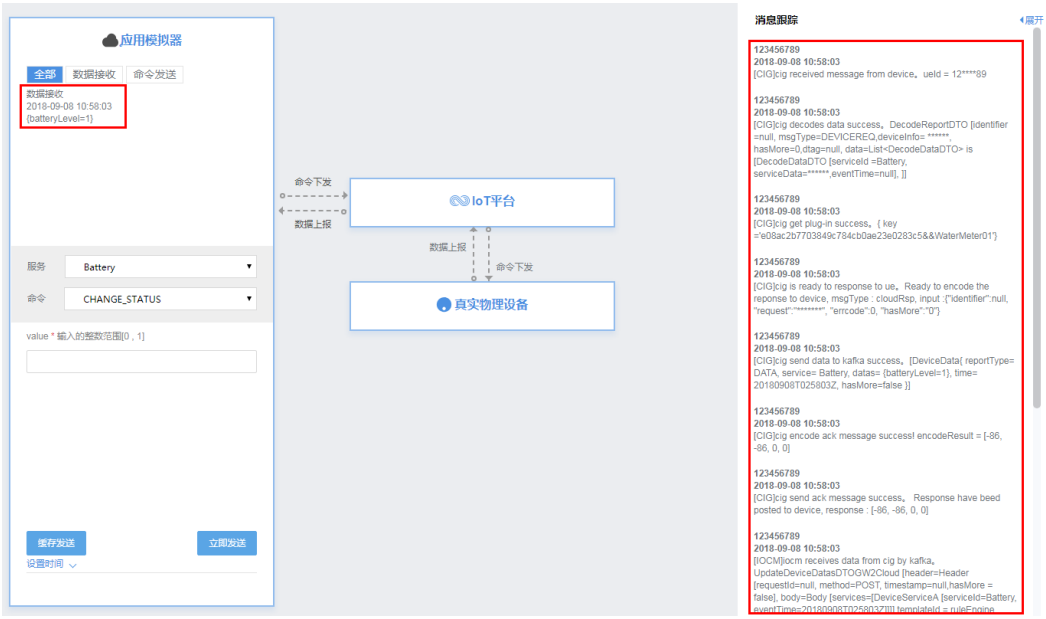
步骤3 系统将弹出“新增测试设备”窗口，勾选“有真实的物理设备”，完成各项参数配置后，点击“创建”。

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

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。

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



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

----结束

使用虚拟设备调测

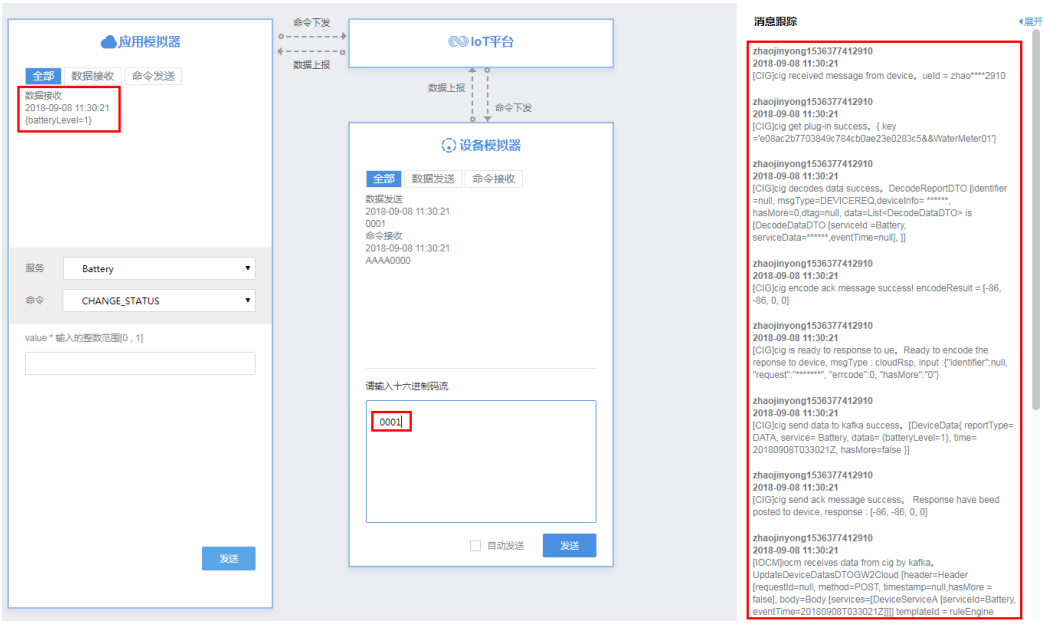
步骤1 在产品开发空间，点击“在线调测”。

步骤2 在“设备列表”区域，点击“新增测试设备”。

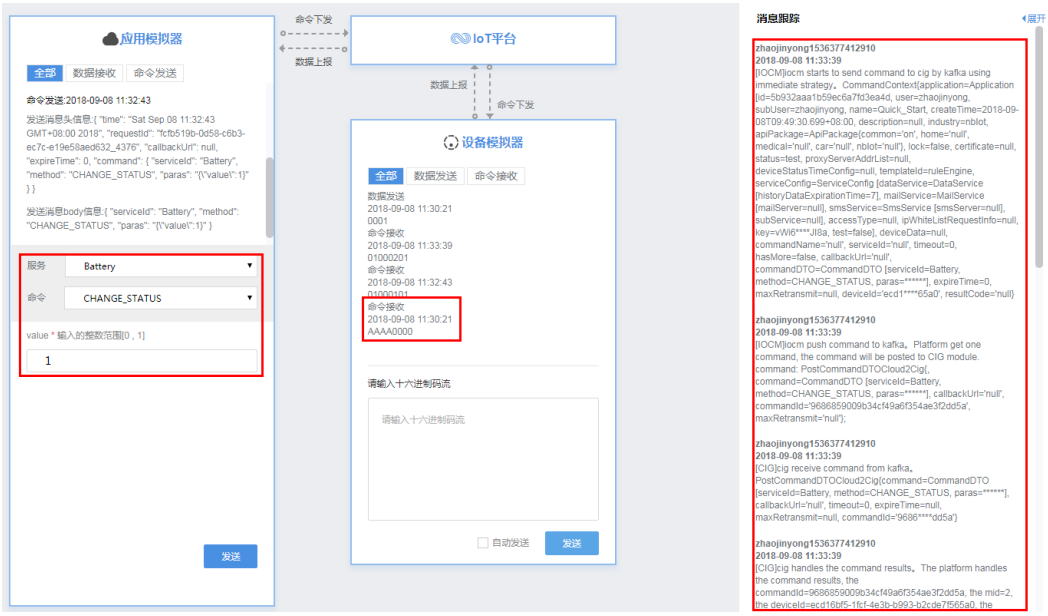
步骤3 系统将弹出“新增测试设备”窗口，勾选“没有真实的物理设备”，点击“创建”。

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”，每款产品下只能够创建一个虚拟设备。

步骤5 在“设备模拟器”区域，输入十六进制码流，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

1.7 自助测试

1.7.1 自助测试指导

概述

自助测试是提供端到端的测试用例，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。

前提条件

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。

操作步骤

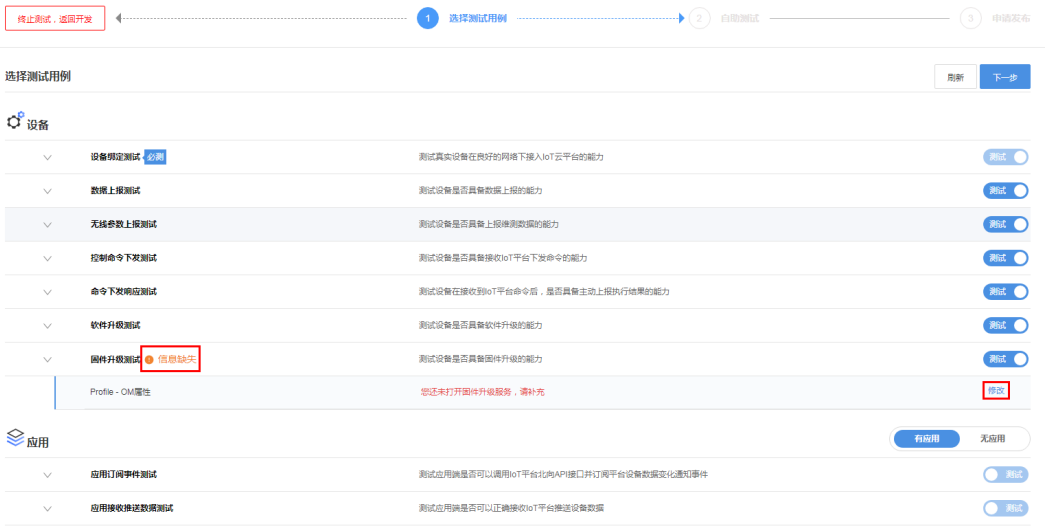
步骤1 在产品开发空间，单击“发起自助测试”。

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。

  说明

  - 正常发起自助测试，除必测的测试用例外，数据上报测试和控制命令下发测试二者必须选择其一。
  - 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

1.7.2 设备绑定测试

概述

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。

操作步骤

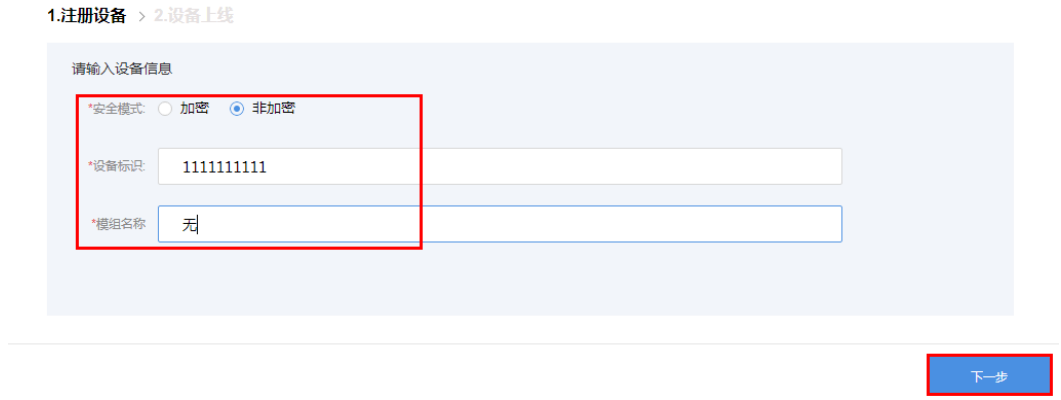
步骤1 在设备绑定测试界面，单击“下一步”，进入注册设备界面。

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”。

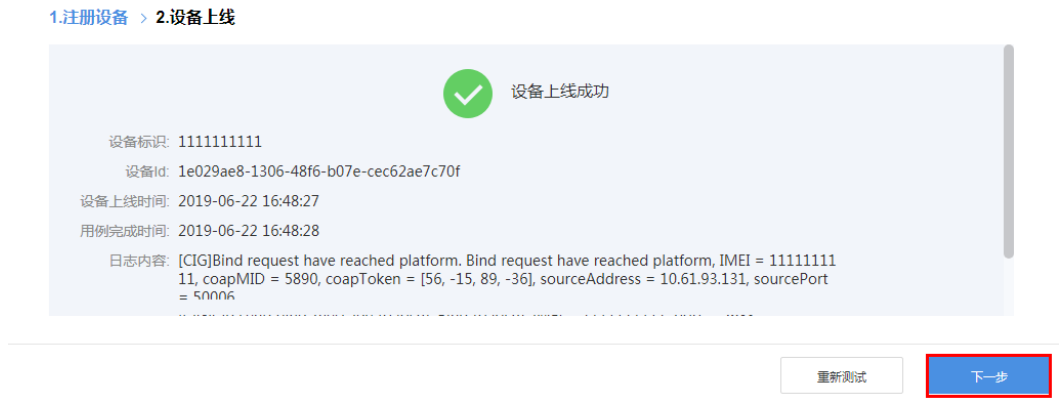
说明

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

1.7.3 数据上报测试

概述

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。

操作步骤

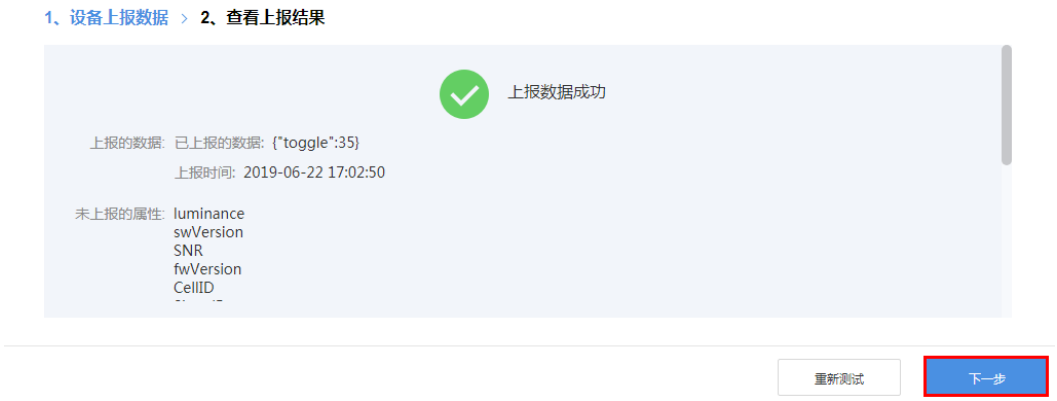
步骤1 在数据上报测试界面，单击“下一步”开始测试。

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，可以点击“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。

说明

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

1.7.4 无线参数上报测试

概述

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。

操作步骤

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

1.7.5 控制命令下发测试

概述

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。

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。

操作步骤

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。

- 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

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，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。

----结束

1.7.6 命令下发响应测试

概述

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。

操作步骤

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。

说明

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。

----结束

1.7.7 固件升级测试

概述

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。

操作步骤

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。

说明

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

1.7.8 软件升级测试

概述

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。

操作步骤

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。

说明

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

1.7.9 应用订阅事件测试

概述

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**。

操作步骤

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

1.7.10 应用接收推送数据测试

概述

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。

操作步骤

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

1.8 产品发布

概述

如果开发中心已经对接产品中心，则开发者可以向产品中心申请发布通过自助测试的产品：可以申请公开发布或私有发布。

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。

步骤3 选择发布方式：“公开发布”或“私有发布”，单击“发布”。

----结束

2 设备对接

2.1 创建应用

概述

在管理门户中创建一个应用，用于将真实的设备和北向应用服务器接入到IoT平台，实现设备的数据采集和设备管理。

创建应用后，IoT平台会分配北向应用服务器和南向设备的接入地址和端口信息，方便用户快速对接应用和设备。

操作指导

步骤1 登录华为云管理控制台，在IoT设备管理服务中，点击“进入管理门户”。

步骤2 选择“应用列表”，单击“创建应用”。

步骤3 参考**表2-1**按照实际情况填写配置参数。

表 2-1 应用参数说明

参数名称	参数说明
基本信息	
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。

参数名称	参数说明
消息跟踪数据授权	设置IoT平台运营管理员可以跟踪发生故障的设备的权限。 ● 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，IoT平台运维管理员跟踪的设备数据保留时间不超过3天。 ● 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。
消息推送	
选择协议	**推送协议** 推送协议是由北向应用服务器在订阅IoT平台的设备信息时设置的传输协议来确定的。北向应用服务器设置的订阅数据推送的传输通道为HTTP时，用户可以设置采用加密的HTTPS协议或者非加密的HTTP协议进行传输数据。 ● HTTPS方式：表示IoT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。 ● HTTP方式：表示IoT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IoT平台与应用服务器之间通信信息泄露风险。 **CA证书** CA证书由应用服务器提供，用于IoT平台对应用服务器进行校验。 **说明** IoT平台已预置了CA证书，该证书仅用于调测使用。在商用场景下，请使用应用服务器侧提供的CA证书。
平台能力	
设备数据处理	IoT平台提供设备历史数据的存储能力，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。 ● 打开开关：即IoT平台会对历史数据进行存储，存储时间以界面显示的存储时间为准。 ● 关闭开关：即IoT平台不对历史数据进行存储。
推送服务	北向应用服务器向IoT平台订阅设备信息，IoT平台能够向北向应用服务器进行消息推送。
其他	
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 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，密钥信息在应用详细页内不可见，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和应用密钥用于北向应用服务器接入IoT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。

步骤5 （可选）对于NB-IoT设备，单击创建的应用，在应用详情的“服务设置”中，设置NB-IoT设备的工作模式，工作模式对应IoT平台下发命令的缓存模式。设置的工作模式请与设备使用的工作模式保持一致。

- 缓存模式：请设置为PSM模式，缓存时间以北向命令下发接口设置的expireTime为准，如果北向接口未设置，则默认缓存时间为48h。
- 立即下发模式：请设置为DRX或eDRX模式，下发的命令不缓存，直接下发。

----结束

2.2 导入产品模型

概述

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

产品模型开发完成并发布后，可以通过管理门户导入在产品中心上发布的产品模型。

操作指导

步骤1 选择“产品模型”，单击“新增产品模型”。

步骤2 产品模型可通过从产品中心导入和从本地导入两种方式。

- 从产品中心导入：
  a. 选择“从产品中心导入”，进入到产品中心页面。
  b. 通过产品名称、设备类型或者厂商名称搜索产品，在搜索结果中单击需要导入的产品名称。
  c. 查看产品是否为公开产品。
     - 公开产品，单击“导入该产品”，即可启动从产品中心导入产品到平台。
     - 非公开产品，需要用户输入验证码校验（验证码请前往产品中心获取）。校验通过后，则可以查看产品详情和导入产品到平台。
- 从本地导入：

a. 选择“本地导入”进入到本地导入产品的页面。
b. 在弹出的窗口中输入产品名称，并上传资源文件。
c. 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。

步骤3 在“产品模型”页面查看导入结果。

- 导入失败：可在“失败原因”中查看导入失败的原因，用户可根据失败原因定位错误。
- 导入成功：可单击“详情”，查看产品模型详情。

说明

用户可以在产品列表中删除不再使用的产品，单击“删除”即可，删除该产品后会导致该产品下的设备功能无法使用，在产品中心重新导入该产品后，该产品下的设备功能恢复正常。

----结束

2.3 注册设备

概述

在IoT平台上注册设备，定义相关的设备参数信息。当后续实际设备接入IoT平台时，设备和平台之间进行鉴权认证，成功后，实际设备接入IoT平台，实现平台和设备的连接和通信。

操作指导

步骤1 选择“ 设备 > 设备注册”。

步骤2 选择页签“单一注册”，单击右上角“创建”，按照如下表格填写参数后，单击“确定”。

表 2-2 单个注册参数说明

参数名称	配置原则
选择产品	选择要注册的产品名称。 只有在“产品模型”里定义了产品，此处才可以选择具体的产品。如没有，请先上传或直接创建产品模型。
设备标识码	设备唯一物理标识，如IMEI、MAC地址等，用于设备在接入IoT平台时携带该标识信息完成接入鉴权。 ● 原生MQTT设备：通过注册成功后生成的“设备ID”（与设备标识码一一对应）和“密钥”接入平台。 ● NB-IoT设备、集成Agent Lite SDK的设备：设备通过注册时填写的“设备标识码”和“预置密钥”接入平台。
预置密钥	● NB-IoT设备、集成Agent Lite SDK的设备接入时，用于设备和IoT平台之间的传输通道安全加密。 ● 原生MQTT设备接入时，无需填写。
确认密钥	再次填写预置密钥。

----结束

2.4 接入设备

概述

将真实设备接入到IoT平台中，并验证设备能够上报数据到IoT平台，并在管理门户中正常显示。

前提条件

已完成设备侧的开发，详情请参考开发设备。

操作指导

步骤1 将设备接入IoT平台的地址和端口信息，更改为IoT设备管理服务的设备对接信息，对接信息可在“管理控制台”中进行查看。

步骤2 （可选）如果是MQTT设备，需要在设备侧加载IoT平台提供的商用CA证书。

步骤3 设备上电，并向IoT平台上报数据。

步骤4 登录管理门户，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IoT平台。

步骤5 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。

说明

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6 在设备详情中的“命令”页签，单击“命令下发”按钮，选择对应的控制命令后，向设备下发控制命令。根据下发的命令，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，且在管理门户中查看下发命令任务的执行结果为“已送达”或“成功”，则表示设备命令下发成功。

说明

- 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IoT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

3 应用对接

3.1 接入应用

概述

将开发的北向应用接入到IoT平台，实现北向应用对设备的远程管理。

前提条件

- 已完成北向应用的开发，详情请参考开发应用。
- 已完成应用创建，详情请参考创建应用。

操作指导

步骤1 将北向应用接入IoT平台的地址和端口信息，更改为IoT设备管理服务的应用对接信息，对接信息可在“管理控制台”中进行查看。

步骤2 将北向应用接入的“应用ID”和“应用密钥”替换为创建应用时分配的信息“应用ID”和“应用密钥”信息。

步骤3 北向应用使用HTTPS协议与IoT平台进行交互，需要将调测证书更换为商用证书。

北向应用访问IoT平台，采用单向认证方式（北向应用认证IoT平台），需要获取IoT平台提供的CA证书，并加载到北向应用中。

步骤4 北向应用通过调用IoT平台的鉴权接口，完成北向应用的接入，鉴权接口信息详见《北向API参考》。

----结束

3.2 订阅数据

概述

应用服务器通过调用IoT平台订阅接口并设置消息推送的回调地址（callbackUrl），告知IoT平台将消息推送到哪里，以及希望推送的通知类型，比如设备业务数据、设备告警等。

订阅推送的协议采用加密传输的HTTPS协议，需要加载对应的证书文件，该证书文件需要由北向应用服务器提供。

操作指导

步骤1 北向应用通过调用IoT平台的订阅接口进行数据订阅，订阅接口信息详见《北向API参考》。

步骤2 登录管理门户，选择“系统管理 > 应用管理 > 应用列表”界面，点击创建的应用。

步骤3 在应用详情的“应用定义”页签中，在消息推送功能中，单击“证书管理”按钮，上传对应的证书文件。

步骤4 单击“添加”按钮，按照表3-1填写相关信息后，单击“确定”完成证书的加载。

表 3-1 加载证书

参数名称	参数说明
CA证书	需要提前申请和购买CA证书文件，CA证书由北向应用服务器提供。
域名/IP与端口	IoT平台推送消息到北向应用服务器的域名或IP地址与端口信息。填写为订阅接口里的callbackurl中对应的域名或IP地址与端口信息。例如：api.ct10649.com:9001或127.0.1.2:8080。
lb昵称	证书加载的LoadBalance对应的昵称，选择默认值：default。
是否检查CNNAME	是否对CA证书的CNNAME进行校验，确保要加载的证书与申请的证书是完全匹配的。建议打开该开关。
CNNAME	“是否检查CNNAME”开关打开时出现。CA证书中携带的证书名称。请向证书申请人员获取。
使用设备证书	在应用服务器需要验证IoT平台的合法性的场景下，打开该开关。 ● 如果关闭该开关，则采用单向认证（IoT平台校验CallBackURL对应的服务端），则CallBackURL对应的服务端也需设置为单向认证。 ● 如果打开该开关，CallBackURL对应的服务端需申请相应的证书文件，并在IoT平台上传设备证书（客户端证书）。
设备证书	设备证书又叫公钥证书，即含有公钥的数字证书。设备证书由北向应用服务器提供。
私钥文件	用户密钥对中的私钥文件。为保护私钥文件的私密性，可以对私钥文件进行加密保护，即只有知道私钥密码才能正常使用私钥。
私钥密码	用于对私钥文件进行加密保护。

----结束

3.3 调测应用

概述

将真实的设备和应用接入到IoT平台后，需要对设备上报数据到IoT平台，IoT平台能正常将设备上报的数据推送给北向应用进行验证；同时，验证北向应用向设备下发命令，设备能正常收到命令并执行成功。

操作指导

步骤1 设备上电，并向IoT平台上报数据。

步骤2 登录管理门户，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IoT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。

说明

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IoT平台推送的数据，如果能正常接收，则表示IoT平台推送消息成功。

步骤5 通过北向应用向设备下发命令，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，且在管理门户中查看下发命令任务的执行结果为“已送达”或“成功”，则表示北向应用向设备下发命令成功。

说明

- 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IoT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

4 设备侧 SDK 使用指南

LiteOS SDK使用指南

4.1 LiteOS SDK 使用指南

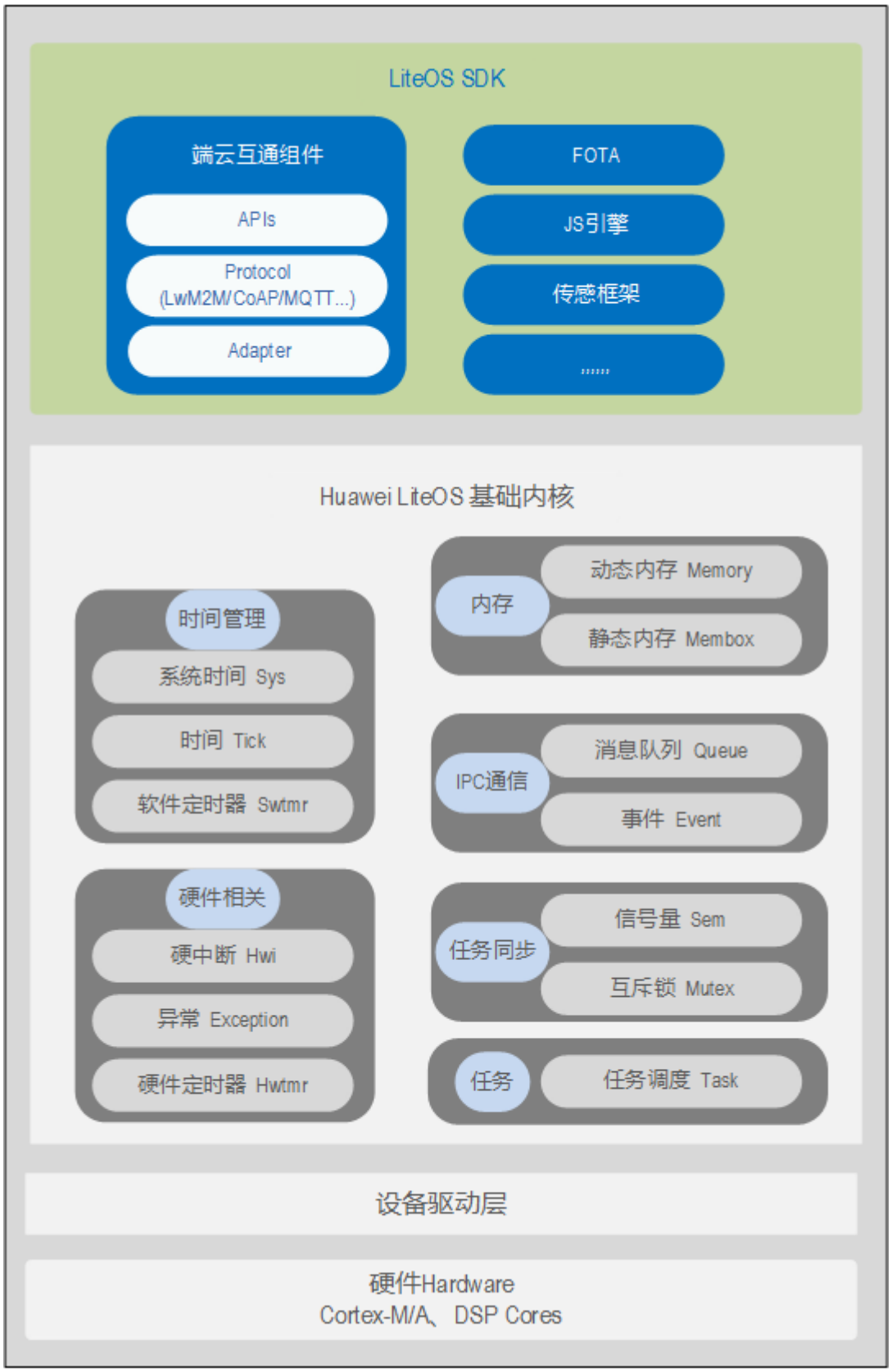
4.1.1 LiteOS SDK 端云互通组件概述

4.1.1.1 背景介绍

LiteOS SDK是Huawei LiteOS软件开发工具包（Software Development Kit），包括端云互通组件、FOTA、JS引擎、传感框架等内容。

本文档介绍的LiteOS SDK包括了LiteOS SDK端云互通组件。端云互通组件是华为物联网解决方案中，资源受限终端对接到 IoT云平台的重要组件。端云互通组件提供端云协同能力，集成了 LwM2M、CoAP、MQTT、mbedTLS、LwIP 等全套 IoT 互联互通协议栈，且在 LwM2M 的基础上，提供端云互通组件开放API，用户只需关注自身的应用，而不必关注 LwM2M 实现细节，直接使用 LiteOS SDK端云互通组件封装的 API，通过四个步骤就能简单快速地实现与华为 OceanConnect IoT平台安全可靠连接。使用 LiteOS SDK端云互通组件，用户可以大大减少开发周期，聚焦自己的业务开发，快速构建自己的产品。

图 4-1 Huawei LiteOS 架构图



4.1.1.2 系统方案

Huawei LiteOS SDK端云互通组件针对“单模组、单MCU”和“外置MCU+模组”两种应用场景，提供了不同的软件架构：

图 4-2 单模组/单 MCU 软件架构

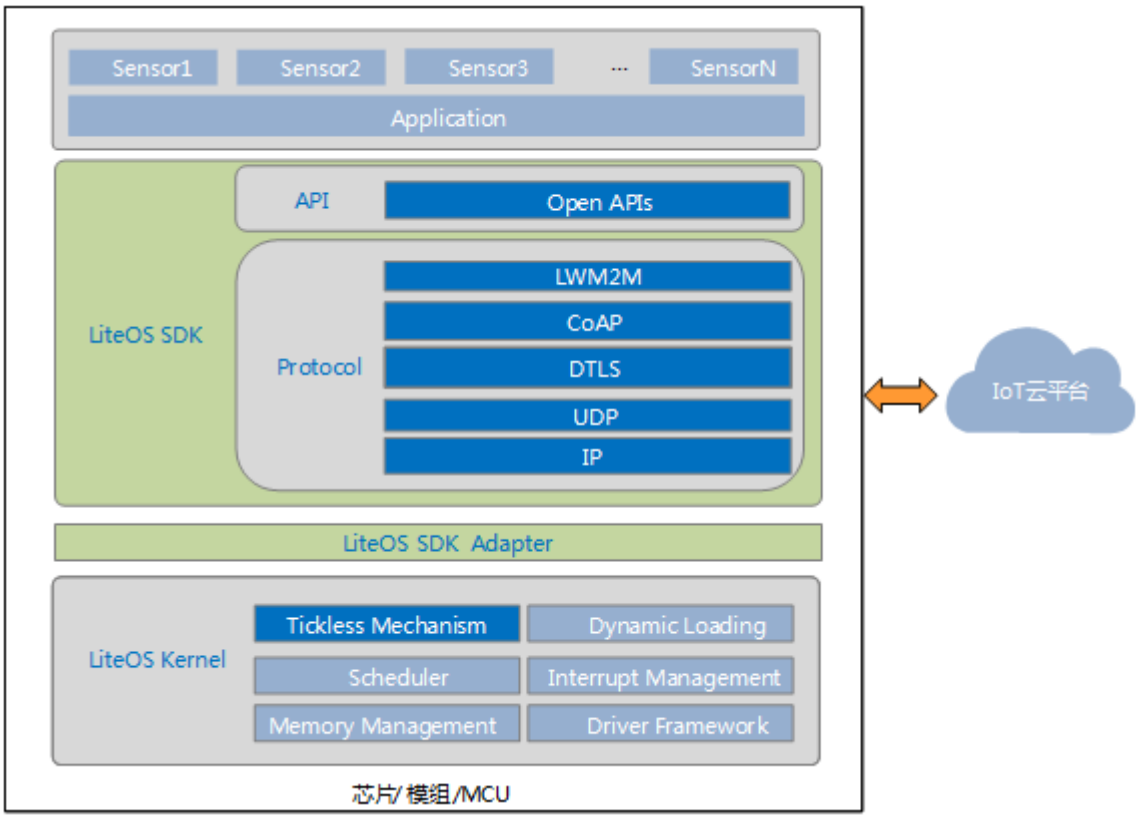
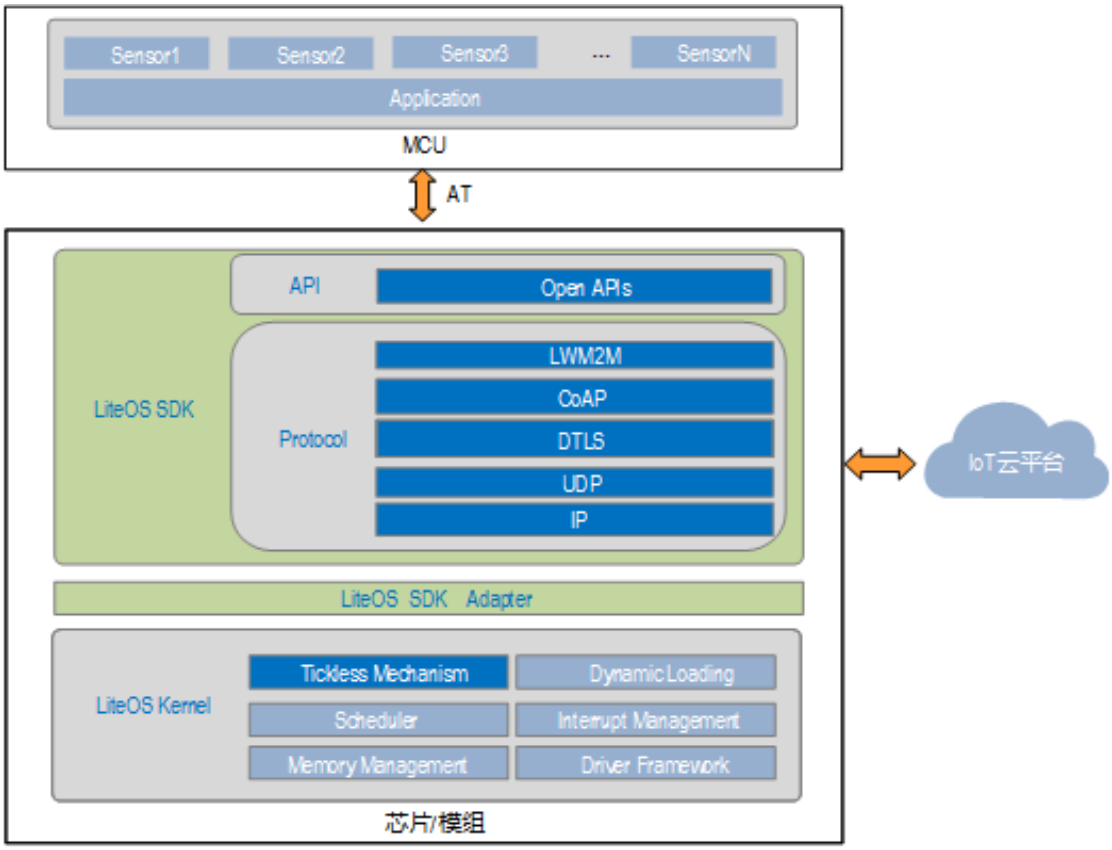


图 4-3 MCU+芯片/模组软件架构



LiteOS SDK 端云互通组件软件主要由三个层次构成：

- **开放API层**：LiteOS SDK端云互通组件的开放API为应用程序定义了通用接口，终端设备调用开放API能快速完成华为OceanConnect IoT平台的接入、业务数据上报、下发命令处理等。对于外置MCU+模组的场景，LiteOS SDK端云互通组件还提供了AT 命令适配层，用于对AT命令做解析。
- **协议层**：LiteOS SDK端云互通组件集成了LwM2M/CoAP/DTLS/TLS/UDP等协议。
- **驱动及网络适配层**：LiteOS SDK端云互通组件为了方便终端设备进行集成和移植，提供了驱动及网络适配层，用户可以基于SDK提供的适配层接口列表，根据具体的硬件平台适配硬件随机数、内存管理、日志、数据存储以及网络Socket等接口。

LiteOS基础内核：为用户终端设备提供RTOS特性。

4.1.1.3 集成策略

4.1.1.3.1 可集成性

LiteOS SDK端云互通组件作为独立的组件，不依赖特定的芯片架构和网络硬件类型，可以轻松地集成到各种通信模组上，如NB-IoT模组、eMTC模组、WIFI模组、GSM模组、以太网硬件等。

4.1.1.3.2 可移植性

LiteOS SDK端云互通组件的Adapter层提供了常用的硬件及网络适配接口，终端或者模组厂家可以根据自己的硬件实现这些接口后，即可完成LiteOS SDK端云互通组件的移植。需要移植的接口列表及相关函数如下：

表 4-1 LiteOS SDK 端云互通组件需要移植适配的接口列表

接口分类	接口名	说明
网络Socket相关接口	atiny_net_connect	创建socket网络连接
	atiny_net_recv	接收函数
	atiny_net_send	发送函数
	atiny_net_recv_timeout	阻塞式接收函数
	atiny_net_close	关闭socket网络连接
硬件相关接口	atiny_gettime_ms	获取系统时间，单位ms
	atiny_usleep	延时函数，单位us
	atiny_random	硬件随机数函数
	atiny_malloc	动态内存申请
	atiny_free	动态内存释放
	atiny_snprintf	格式化字符串
	atiny_printf	日志输出
资源互斥相关接口	atiny_mutex_create	创建互斥锁
	atiny_mutex_destroy	销毁互斥锁
	atiny_mutex_lock	获取互斥锁
	atiny_mutex_unlock	释放互斥锁

说明

LiteOS SDK端云互通组件支持OS方式移植，也支持无OS方式移植，推荐使用支持OS方式移植。

LiteOS SDK端云互通组件支持固件升级，需要适配atiny_storage_devcie_s对象，供组件使用。

```
atiny_storage_devcie_s *atiny_get_hal_storage_device(void);

struct atiny_storage_device_tag_s;
struct atiny_storage_device_tag_s;
typedef struct atiny_storage_device_tag_s  atiny_storage_device_s;
struct atiny_storage_device_tag_s
{
   //设备初始化
   int (*init)( storage_device_s *this);
   //准备开始写
   int (*begin_software_download)( storage_device_s *this);
   //写软件，从offset写，buffer为内容，长度为len
```

```
    int (*write_software)( storage_device_s *this , uint32_t offset, const char *buffer, uint32_t
len);

    //下载结束
    int (*end_software_download)( storage_device_s *this);
    //激活软件
    int (*active_software)( storage_device_s *this);
    //得到激活的结果, 0成功，1失败
    int (*get_active_result)( storage_device_s *this);
    //写update_info, 从offset写，buffer为内容，长度为len
    int (*write_update_info)( storage_device_s *this, long offset, const char *buffer, uint32_t
len);
    //读update_info, 从offset写，buffer为内容，长度为len
    int (*read_update_info)( storage_device_s *this, long offset, char *buffer, uint32_t len);
};
```

4.1.1.3.3 集成约束

LiteOS SDK端云互通组件集成需要满足一定的硬件规格：

- 要求模组/芯片有物理网络硬件支持，能支持UDP协议栈。
- 模组/芯片有足够的Flash和RAM资源供LiteOS SDK端云互通组件协议栈做集成。建议的硬件选型规格如下表所示：

表 4-2 硬件选型规格建议

RAM	Flash
>32K	>128K

说明

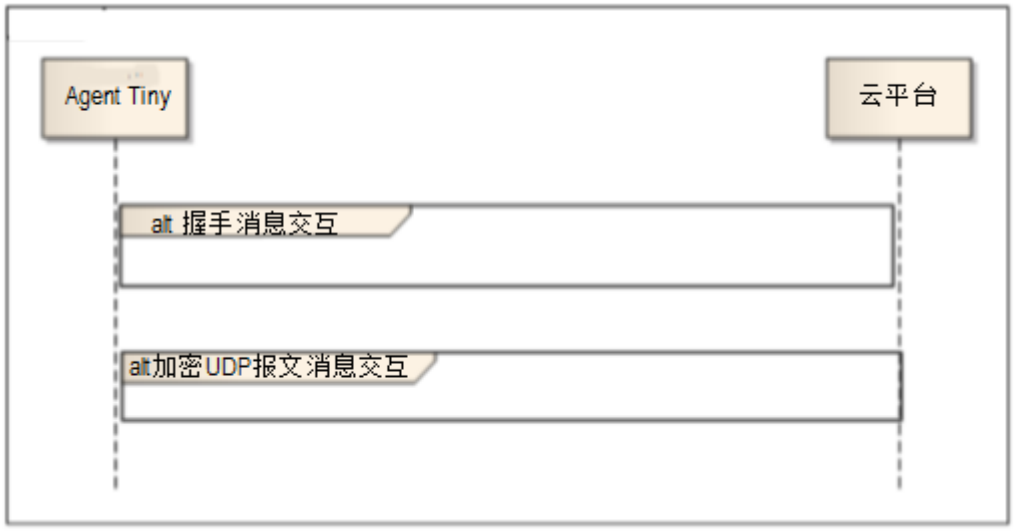
推荐的硬件选型规格考虑LiteOS SDK端云互通组件本身占用的资源（开放API+物联网协议栈+安全协议+SDK驱动及网络适配层），也考虑用户业务demo的最小实现占用的资源（芯片驱动程序、传感器驱动程序、基本业务流程等）。该规格仅为推荐规格，具体选型需要用户根据自身业务再做评估。

4.1.1.4 安全

LiteOS SDK端云互通组件支持DTLS(Datagram Transport Layer Security)，即数据包传输层安全性协议。目前支持PSK(Pre-Shared Keys)预共享密钥模式，后续会扩展支持其他模式。

LiteOS SDK端云互通组件首先和物联网开放平台完成握手流程，后续的应用数据将全部为加密数据，如图所示：

图 4-4 DTLS 协议交流互动

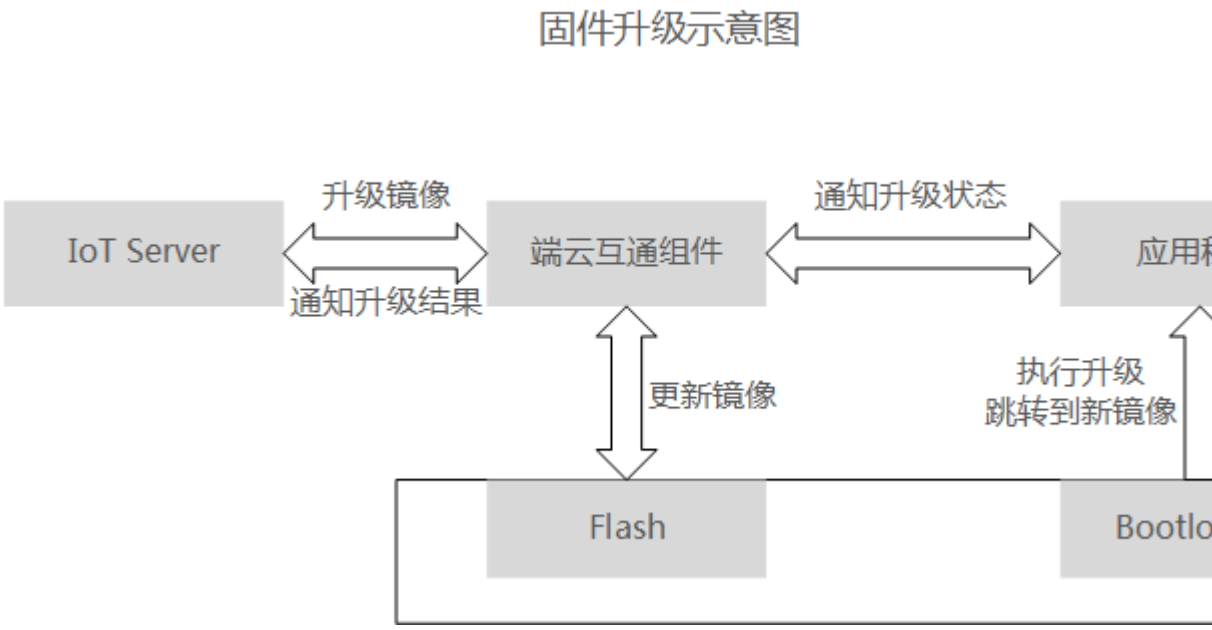


4.1.1.5 升级

LiteOS SDK端云互通组件支持物联网开放平台的远程固件升级，且具备断点续传、固件包完整性保护等特性。

固件升级功能和流程如图所示：

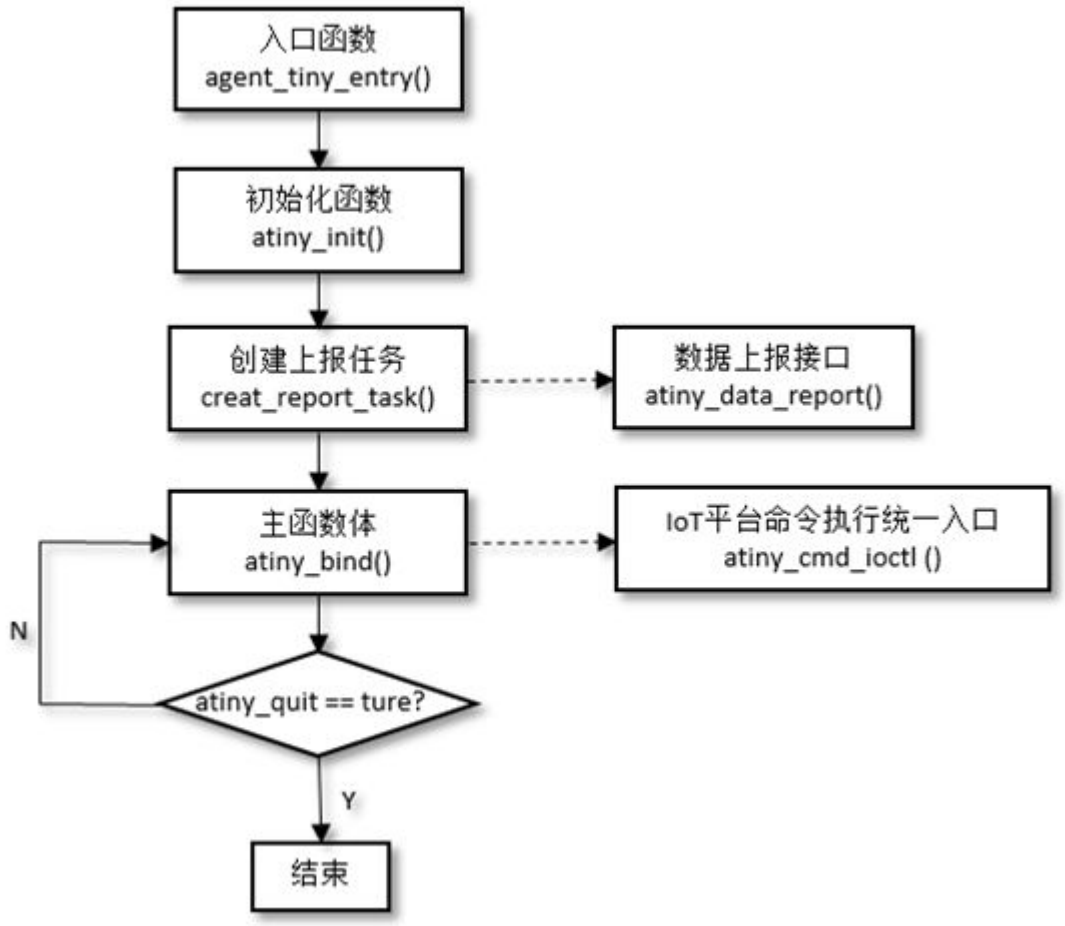
图 4-5 固件升级功能示意图



4.1.2 端侧对接流程

设备接入IoT平台后，IoT平台才可以对设备进行管理。设备接入平台时，需要保证IoT平台已经有对应应用，并且已经在该应用下注册了此设备。本节介绍端侧设备是如何通过端云互通组件与IoT平台实现对接的。首先给出端侧设备对接IoT平台的整体示意图。

图 4-6 端侧设备对接 IoT 平台的整体示意图



本小节将根据上图所示的流程，向开发者介绍终端设备是如何一步步地接入IoT平台，并进行数据上报与命令执行的。

4.1.2.1 环境准备

在开发之前，需要提前获取如下信息：

- Huawei LiteOS及LiteOS SDK源代码。工程整体结构如下。

```
├─── arch //架构相关文件
|  ├─── arm
|  └─── msp430
├─── build
|  └─── Makefile
├─── components //LiteOS各类组件
|  ├─── connectivity
|  ├─── fs
```

```
|   ├──── lib
|   ├──── log
|   ├──── net
|   ├──── ota
|   └──── security
├──── demos //示例程序
|   ├──── agenttiny_lwm2m //本章中列出的所有示例程序，均来自该目录下的agent_tiny_demo.c文件
|   ├──── agenttiny_mqtt
|   ├──── dtls_server
|   ├──── fs
|   ├──── kernel
|   └──── nbiot_without_atiny
├──── doc //说明文档
|   ├──── Huawei_LiteOS_Developer_Guide_en.md
|   ├──── Huawei_LiteOS_Developer_Guide_zh.md
|   ├──── Huawei_LiteOS_SDK_Developer_Guide.md
|   ├──── LiteOS_Code_Info.md
|   ├──── LiteOS_Commit_Message.md
|   ├──── LiteOS_Contribute_Guide_GitGUI.md
|   ├──── LiteOS_Supported_board_list.md
|   └──── meta
├──── include //工程需要的头文件
|   ├──── at_device
|   ├──── at_frame
|   ├──── atiny_lwm2m
|   ├──── atiny_mqtt
|   ├──── fs
|   ├──── log
|   ├──── nb_iot
|   ├──── osdepends
|   ├──── ota
|   ├──── sal
```

```
|   └──── sota
├──── kernel //系统内核
|   ├──── base
|   ├──── extended
|   ├──── include
|   ├──── los_init.c
|   └──── Makefile
├──── LICENSE //许可
├──── osdepends //依赖项
|   └──── liteos
├──── README.md
├──── targets //BSP工程
|   ├──── Cloud_STM32F429IGTx_FIRE
|   ├──── Mini_Project
|   ├──── NXP_LPC51U68
|   └──── STM32F103VET6_NB_GCC
└──── tests //测试用例
├──── cmockery
├──── test_agenttiny
├──── test_main.c
├──── test_sota
└──── test_suit
```

源代码托管在GitHub，地址为**https://github.com/LiteOS/LiteOS**。

- 集成开发工具:
  - MDK 5.18版本或者以上版本，从MDK官方网站下载。
  - MDK依赖的pack包

    说明

    MDK工具需要license，请从MDK官方获取。

4.1.2.2 LiteOS SDK 端云互通组件入口函数

使用LiteOS SDK端云互通组件agent tiny对接IoT平台，首先需要一个入口函数agent_tiny_entry()。

接口名	描述

void agent_tiny_entry(void)	LiteOS SDK端云互通组件的入口函数。该接口将进行agent tiny的初始化相关操作，创建上报任务，并调用agent tiny主函数体。 参数列表：空 返回值：空

开发者可以通过LiteOS内核提供的任务机制，创建一个主任务main_task。在主任务中调用入口函数agent_tiny_entry()，开启agent tiny工作流程。

```
UINT32 creat_main_task()
 {
     UINT32 uwRet = LOS_OK;
     TSK_INIT_PARAM_S task_init_param;
     task_init_param.usTaskPrio = 0;
     task_init_param.pcName = "main_task";
     task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)main_task;
     task_init_param.uwStackSize = 0x1000;
     uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);
     if(LOS_OK != uwRet)
     {
         return uwRet;
     }
     return uwRet;
 }
```

4.1.2.3 LiteOS SDK 端云互通组件初始化

在入口函数中，需要调用atiny_init()进行agent tiny的初始化相关操作。

接口名	描述
int atiny_init(atiny_param_t* atiny_params, void** phandle)	LiteOS SDK端云互通组件的初始化接口，由LiteOS SDK端云互通组件实现，设备调用。 参数列表：参数atiny_params为入参，包含初始化操作所需的各个变量，具体请参考服务器参数结构体atiny_param_t；参数phandle为出参，表示当前创建的agent tiny的句柄。 返回值：整形变量，标识初始化成功或失败的状态。

对于入参atiny_params的设定，要根据具体的业务来进行。开发者可以参考下面的代码。

```
#ifdef CONFIG_FEATURE_FOTA
     hal_init_ota();   //若定义FOTA功能，则需进行FOTA相关初始化
 #endif

 #ifdef WITH_DTLS
     device_info->endpoint_name = g_endpoint_name_s;  //加密设备验证码
 #else
     device_info->endpoint_name = g_endpoint_name;    //非加密设备验证码
```

```
#endif
#ifdef CONFIG_FEATURE_FOTA
    device_info->manufacturer = "Lwm2mFota";    //设备厂商
    device_info->dev_type = "Lwm2mFota";        //设备类型
#else
    device_info->manufacturer = "Agent_Tiny";
#endif
    atiny_params = &g_atiny_params;
    atiny_params->server_params.binding = "UQ";   //绑定方式
    atiny_params->server_params.life_time = 20;   //生命周期
    atiny_params->server_params.storing_cnt = 0;  //缓存数据报文个数

    atiny_params->server_params.bootstrap_mode = BOOTSTRAP_FACTORY;   //引导模式
    atiny_params->server_params.hold_off_time = 10;    //等待时延

    //pay attention: index 0 for iot server, index 1 for bootstrap server.
    iot_security_param = &(atiny_params->security_params[0]);
    bs_security_param = &(atiny_params->security_params[1]);

    iot_security_param->server_ip = DEFAULT_SERVER_IPV4;  //服务器地址
    bs_security_param->server_ip = DEFAULT_SERVER_IPV4;

#ifdef WITH_DTLS
    iot_security_param->server_port = "5684";   //加密设备端口号
    bs_security_param->server_port = "5684";

    iot_security_param->psk_Id = g_endpoint_name_iots;          //加密设备验证码
    iot_security_param->psk = (char *)g_psk_iot_value;          //PSK码
    iot_security_param->psk_len = sizeof(g_psk_iot_value);      //PSK码长度

    bs_security_param->psk_Id = g_endpoint_name_bs;
    bs_security_param->psk = (char *)g_psk_bs_value;
    bs_security_param->psk_len = sizeof(g_psk_bs_value);
#else
    iot_security_param->server_port = "5683";    //非加密设备端口号
    bs_security_param->server_port = "5683";

    iot_security_param->psk_Id = NULL;    //非加密设备，无需PSK相关参数设置
    iot_security_param->psk = NULL;
    iot_security_param->psk_len = 0;

    bs_security_param->psk_Id = NULL;
    bs_security_param->psk = NULL;
    bs_security_param->psk_len = 0;
#endif
```

设定好atiny_params后，即可根据设定的参数对agent tiny进行初始化。

```
  if(ATINY_OK != atiny_init(atiny_params, &g_phandle))
   {
       return;
   }
```

对于初始化接口atiny_init()内部，主要进行入参合法性的检验，agent tiny所需资源的创建等工作，一般不需要开发者进行修改。

4.1.2.4 创建数据上报任务

在完成agent tiny的初始化后，需要通过调用creat_report_task()创建一个数据上报的任务app_data_report()。

```
   UINT32 creat_report_task()
    {
        UINT32 uwRet = LOS_OK;
        TSK_INIT_PARAM_S task_init_param;
        UINT32 TskHandle;
```

```
    task_init_param.usTaskPrio = 1;
    task_init_param.pcName = "app_data_report";
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)app_data_report;
    task_init_param.uwStackSize = 0x400;
    uwRet = LOS_TaskCreate(&TskHandle, &task_init_param);
    if(LOS_OK != uwRet)
    {
        return uwRet;
    }
    return uwRet;
}
```

在app_data_report()中应该完成对数据上报数据结构data_report_t的赋值，包括数据缓冲区地址buf，收到平台ack响应后的回调函数callback，数据cookie，数据长度len，以及数据上报类型type（在这里固定为APP_DATA）。

```
uint8_t buf[5] = {0, 1, 6, 5, 9};
 data_report_t report_data;
 int ret = 0;
 int cnt = 0;
 report_data.buf = buf;
 report_data.callback = ack_callback;
 report_data.cookie = 0;
 report_data.len = sizeof(buf);
 report_data.type = APP_DATA;
```

完成对report_data的赋值后，即可通过接口atiny_data_report()上报数据。

接口名	描述
int atiny_data_report(void* phandle, data_report_t* report_data)	LiteOS SDK端云互通组件数据上报接口，由LiteOS SDK端云互通组件实现，设备调用，设备应用数据使用该接口上报。该接口为阻塞接口，不允许在中断中使用。 参数列表：参数phandle为调用初始化接口atiny_init()得到的agent tiny的句柄；参数report_data为数据上报数据结构。 返回值：整形变量，标识数据上报成功或失败的状态。

示例代码中的上报任务实现方法如下。

```
while(1)
 {
    report_data.cookie = cnt;
    cnt++;
    ret = atiny_data_report(g_phandle, &report_data);   //数据上报接口
    ATINY_LOG(LOG_DEBUG, "data report ret: %d\n", ret);
    (void)LOS_TaskDelay(250 * 8);
 }
```

4.1.2.5 LiteOS SDK 端云互通组件命令处理接口

IoT平台下发的各类命令，都通过接口atiny_cmd_ioctl()来具体执行。

接口名	描述

int atiny_cmd_ioctl (atiny_cmd_e cmd, char* arg, int len);	LiteOS SDK端云互通组件申明和调用，由开发者实现。该接口是LwM2M标准对象向设备下发命令的统一入口。 参数列表：参数cmd为具体命令字，比如下发业务数据，下发复位，升级命令等；参数arg为存放命令参数的缓存；参数len为缓存大小。 返回值：空。

atiny_cmd_ioctl()是LiteOS SDK端云互通组件定义的一个通用可扩展的接口，其命令字如atiny_cmd_e所定义，用户根据自身需求进行选择性实现，也可以根据自身需求进行扩展。常用的接口定义如下表所示，每一个接口都和atiny_cmd_e的枚举值一一对应：

回调接口函数	描述
int atiny_get_manufacturer(char* manufacturer,int len)	获取厂商名字，参数manufacturer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的厂商名字，长度不能超过参数len。
int atiny_get_dev_type(char * dev_type,int len)	获取设备类型，参数dev_type指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备类型，长度不能超过参数len。
int atiny_get_model_number((char * model_numer, int len)	获取设备模型号，参数model_numer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备模型号，长度不能超过参数len。
int atiny_get_serial_number(char* num,int len)	获取设备序列号，参数numer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备序列号，长度不能超过参数len。
int atiny_get_dev_err(int* arg，int len)	获取设备状态，比如内存耗尽、电池不足、信号强度低等，参数arg由LiteOS SDK端云互通组件分配，用户填充，长度不能超过len。
int atiny_do_dev_reboot(void)	设备复位。
int atiny_do_factory_reset(void)	厂商复位。
int atiny_get_baterry_level(int* voltage)	获取电池剩余电量。
int atiny_get_memory_free(int* size)	获取空闲内存大小。
int atiny_get_total_memory(int* size)	获取总共内存大小。
int atiny_get_signal_strength(int* singal_strength)	获取信号强度。
int atiny_get_cell_id(long* cell_id)	获取小区ID。

int atiny_get_link_quality(int* quality)	获取信道质量。
int atiny_write_app_write(void* user_data, int len)	业务数据下发。
int atiny_update_psk(char* psk_id, int len)	预置共享密钥更新。

其中，开发者需要根据自身的业务，在接口atiny_write_app_write()中实现自己的命令响应。

```
int atiny_write_app_write(void* user_data, int len)
 {
     (void)atiny_printf("write num19 object success\r\n");
     return ATINY_OK;
 }
```

4.1.2.6 LiteOS SDK 端云互通组件主函数体

完成了数据上报任务的创建与命令处理接口的实现，agent tiny进入到对接IoT平台的核心步骤atiny_bind()。

接口名	描述
int atiny_bind(atiny_device_info_t* device_info, void* phandle)	LiteOS SDK端云互通组件的主函数体，由LiteOS SDK端云互通组件实现，设备调用，调用成功后，不会返回。该接口是LiteOS SDK端云互通组件主循环体，实现了LwM2M协议处理，注册状态机，重传队列，订阅上报。 参数列表：参数device_info为终端设备参数结构体；参数phandle为调用初始化接口atiny_init()得到的agent tiny的句柄。 返回值：整形变量，标识LiteOS SDK端云互通组件主函数体执行的状态。只有执行失败或者调用了LiteOS SDK端云互通组件去初始化接口atiny_deinit()才会返回。

atiny_bind()会根据LwM2M协议标准，进行LwM2M客户端创建与注册，并将数据上报任务app_data_report()中上报的数据递交给通信模块发送到IoT平台，同时接受IoT平台下发的命令消息，解析后由命令处理接口atiny_cmd_ioctl()统一进行处理。与atiny_init()一样，atiny_bind()内部一般不需要开发者进行修改。

说明：关于LwM2M协议相关内容，请开发者参考附录。

LiteOS SDK端云互通组件通过主函数体，不断地进行数据上报与命令处理。当调用LiteOS SDK端云互通组件去初始化接口atiny_deinit()时，退出主函数体。

接口名	描述

void atiny_deinit(void* phandle);	LiteOS SDK端云互通组件的去初始化接口，由LiteOS SDK端云互通组件实现，设备调用。该接口为阻塞式接口，调用该接口时，会直到agent tiny主任务退出，资源释放完毕，该接口才会退出。 参数列表：参数phandle为调用atiny_init()获取到的LiteOS SDK端云互通组件句柄。 返回值：空

4.1.2.7 数据结构介绍

平台下发命令枚举类型

```
typedef enum
{
    ATINY_GET_MANUFACTURER,          /*获取厂商名字*/
    ATINY_GET_MODEL_NUMBER,          /*获取设备模型，由厂商定义和使用*/
    ATINY_GET_SERIAL_NUMBER,         /*获取设备序列号*/
    ATINY_GET_FIRMWARE_VER,          /*获取固件版本号*/
    ATINY_DO_DEV_REBOOT,             /*下发设备复位命令*/
    ATINY_DO_FACTORY_RESET,          /*厂商复位*/
    ATINY_GET_POWER_SOURCE,          /*获取电源*/
    ATINY_GET_SOURCE_VOLTAGE,        /*获取设备电压*/
    ATINY_GET_POWER_CURRENT,         /*获取设备电流*/
    ATINY_GET_BATERRY_LEVEL,         /*获取电池剩余电量*/
    ATINY_GET_MEMORY_FREE,           /*获取空闲内存*/
    ATINY_GET_DEV_ERR,               /*获取设备状态，比如内存耗尽、电池不足等*/
    ATINY_DO_RESET_DEV_ERR,          /*获取设备复位状态*/
    ATINY_GET_CURRENT_TIME,          /*获取当前时间*/
    ATINY_SET_CURRENT_TIME,          /*设置当前时间*/
    ATINY_GET_UTC_OFFSET,            /*获取UTC时差*/
    ATINY_SET_UTC_OFFSET,            /*设置UTC时差*/
    ATINY_GET_TIMEZONE,              /*获取时区*/
    ATINY_SET_TIMEZONE,              /*设置时区*/
    ATINY_GET_BINDING_MODES,         /*获取绑定模式*/
    ATINY_GET_FIRMWARE_STATE,        /*获取固件升级状态*/
    ATINY_GET_NETWORK_BEARER,        /*获取网络通信承载类型，比如GSM、WCDMA等*/
    ATINY_GET_SIGNAL_STRENGTH,       /*获取网络信号强度*/
    ATINY_GET_CELL_ID,               /*获取网络小区ID*/
    ATINY_GET_LINK_QUALITY,          /*获取网络链路质量*/
    ATINY_GET_LINK_UTILIZATION,      /*获取网络链路利用率*/
    ATINY_WRITE_APP_DATA,            /*业务数据下发命令字*/
    ATINY_UPDATE_PSK,                /*更新psk命令字*/
    ATINY_GET_LATITUDE,              /*获取设备所处纬度*/
    ATINY_GET_LONGITUDE,             /*获取设备所处经度*/
    ATINY_GET_ALTITUDE,              /*获取设备所处高度*/
    ATINY_GET_SPEED,                 /*获取设备运行速度*/
    ATINY_GET_TIMESTAMP,             /*获取时间戳*/
} atiny_cmd_e;
```

关键事件枚举类型

该枚举类型用于LiteOS SDK端云互通组件把自身状态通知用户

```
typedef enum
{
    ATINY_REG_OK,               /*设备注册成功*/
    ATINY_REG_FAIL,             /*设备注册失败*/
    ATINY_DATA_SUBSCRIBLE,      /*数据开始订阅，设备侧允许上报数据 */
    ATINY_DATA_UNSUBSCRIBLE,    /*数据取消订阅，设备侧停止上报数据*/
    ATINY_FOTA_STATE            /*固件升级状态*/
} atiny_event_e;
```

LwM2M 协议参数结构体

```
typedef struct
 {
     char* binding;                                /*目前支持U或者UQ*/
     int   life_time;                              /*LwM2M协议生命周期，默认50000*/
     unsigned int  storing_cnt;                    /*LwM2M缓存区总字节个数*/
 } atiny_server_param_t;
```

安全及服务器参数结构体

```
typedef struct
 {
     bool  is_bootstrap;      /*是否bootstrap服务器*/
     char* server_ip;         /*服务器ip，字符串表示，支持ipv4和ipv6*/
     char* server_port;       /*服务器端口号*/
     char* psk_Id;            /*预置共享密钥ID*/
     char* psk;               /*预置共享密钥*/
     unsigned short psk_len;  /*预置共享密钥长度*/
 } atiny_security_param_t;
```

上报数据的枚举类型

用户上报数据的数据类型，用户根据自身应用扩展

```
typedef enum
 {
     FIRMWARE_UPDATE_STATE = 0,   /*设备固件升级状态*/
     APP_DATA                     /*用户数据*/
 } atiny_report_type_e;
```

服务器参数结构体

```
typedef struct
 {
     atiny_server_param_t   server_params;
     atiny_security_param_t security_params[2];  /*支持一个IOT服务器，一个bootstrap服务器*/
 } atiny_param_t;
```

终端设备参数结构体

```
typedef struct
 {
     char* endpoint_name;    /*北向申请产生的设备标识码*/
     char* manufacturer;     /*北向申请产生的厂商名称*/
     char* dev_type;         /*北向申请产生的设备类型*/
 } atiny_device_info_t;
```

数据上报数据结构

以下枚举值，表述了用户上报的数据，最终的反馈类型，比如数据发送成功，数据发送但未得到确认，具体定义如下：

```
typedef enum
 {
     NOT_SENT = 0,         /*待上报的数据未发送*/
     SENT_WAIT_RESPONSE,   /*待上报的数据已发送，等待响应*/
     SENT_FAIL,            /*待上报的数据发送失败*/
     SENT_TIME_OUT,        /*待上报的数据已发送，等待响应超时*/
     SENT_SUCCESS,         /*待上报的数据发送成功*/
     SENT_GET_RST,         /*待上报的数据已发送，但对端响应RST报文*/
     SEND_PENDING,         /*待上报的数据等待发送*/
 } data_send_status_e;
```

//用户使用以下数据结构上报数据：

```
typedef struct _data_report_t
{
    atiny_report_type_e type;     /*数据上报类型，比如业务数据，电池剩余电量等 */
    int cookie;                   /*数据cookie,用以在ack回调中，区分不同的数据*/
    int len;                      /*数据长度，不应大于MAX_REPORT_DATA_LEN*/
    uint8_t* buf;                 /*数据缓冲区首地址*/
    atiny_ack_callback callback; /*ack回调，其入参取值data_send_status_e类型 */
} data_report_t;
```

4.1.3 LiteOS 快速上云指导

4.1.3.1 设备开发指导

IoT Studio工具支持自动生成代码框架，提供了代码编辑、编译、烧录及调试等一站式开发体验。本节主要介绍基于IoT Studio，将程序样例烧录进小熊派开发板的详细操作步骤。

操作步骤

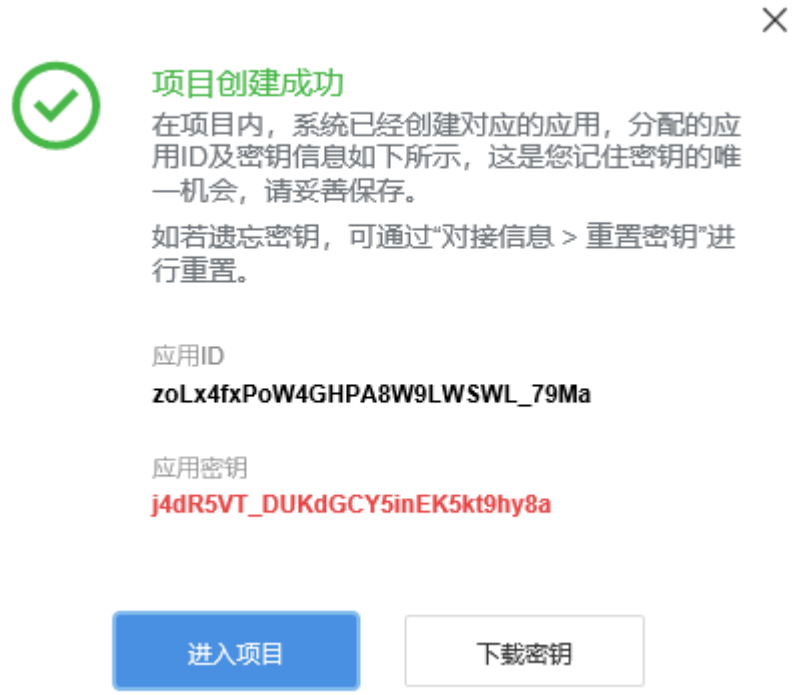
步骤1 访问IoT设备管理服务首页，单击进入“开发中心”。

步骤2 在开发中心首页，单击“新建项目”，然后填写项目名称和选择所属行业后，单击“确定”。

步骤3 创建项目完成后，会弹出“项目创建成功”提示框，请将应用ID和应用密钥下载到本地进行保存，然后点击“进入项目”。

说明

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 进入项目空间首页后，在“产品 > 产品开发”界面中，单击“新建产品”。

步骤5 创建新产品有两种方式，一是基于系统预置的模板进行创建，另一种是自定义一款新产品。本示例以“自定义产品”为例进行说明。

步骤6 在自定义产品页签中，点击“自定义产品”。

步骤7 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“所属行业”等信息后，点击“创建”。

步骤8 在“产品开发”界面选择上述操作步骤新建的产品，进入该产品的开发空间。

步骤9 在产品开发空间，单击“Profile定义”，然后单击“新建服务”。

步骤10 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。

- 下面以添加“LED”服务为例，介绍如何添加命令。用户可根据自己的实际需求参照此步骤添加其他命令。

  a. 在服务名称输入框中输入“LED”，单击 ✓ 。

  b. 单击“命令列表”后的“添加命令”。在弹出窗口中配置“命令名称”，单击“确定”。

  c. 单击“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，单击“确定”。

d. 单击“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，单击“确定”。

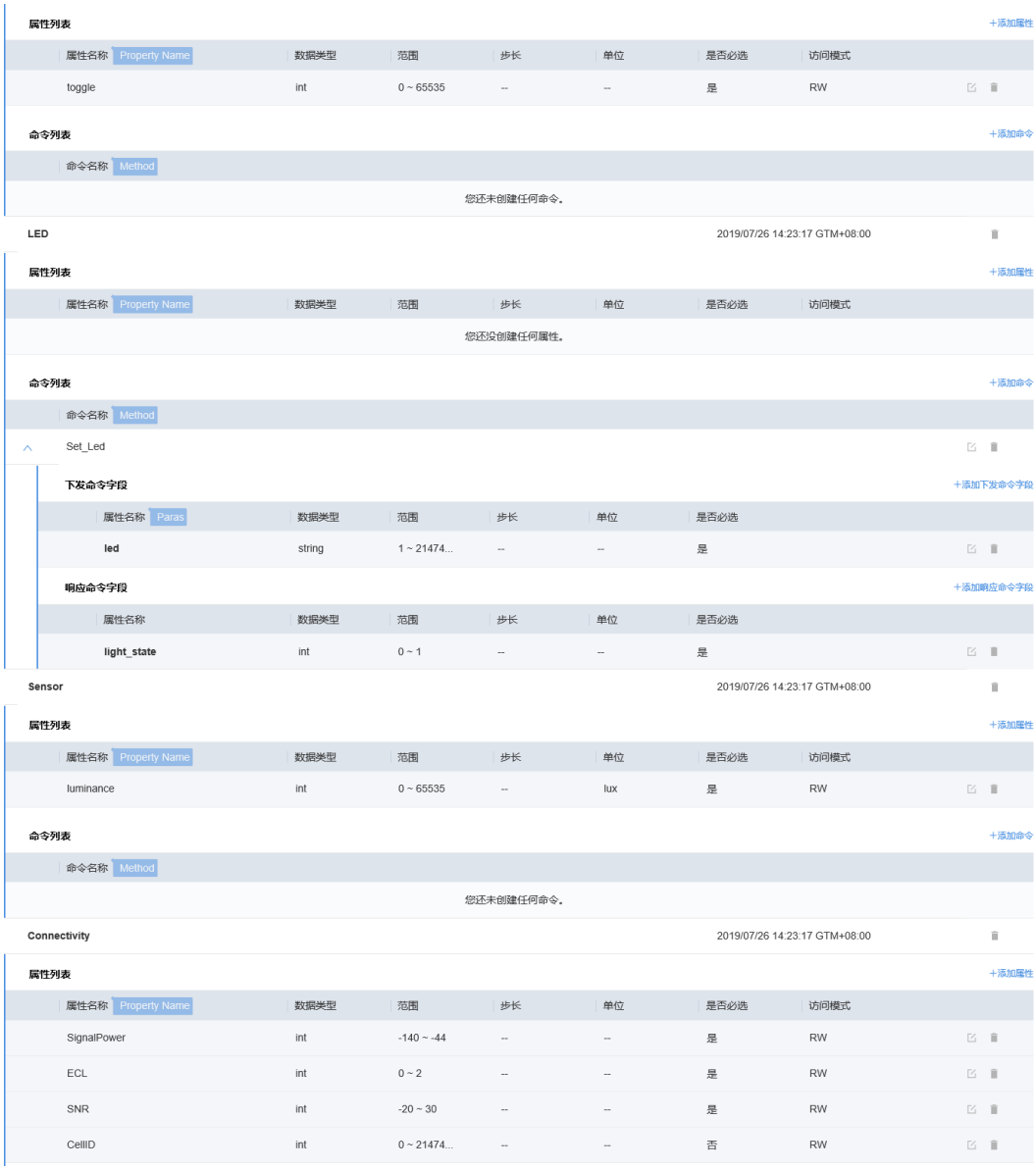
- 下面以添加“Connectivity”服务为例，介绍如何添加属性。用户可参照此步骤添加“Button”以及“Sensor”服务。

  a. 单击“新建服务”，在服务名称输入框中输入“Connectivity”，单击 。

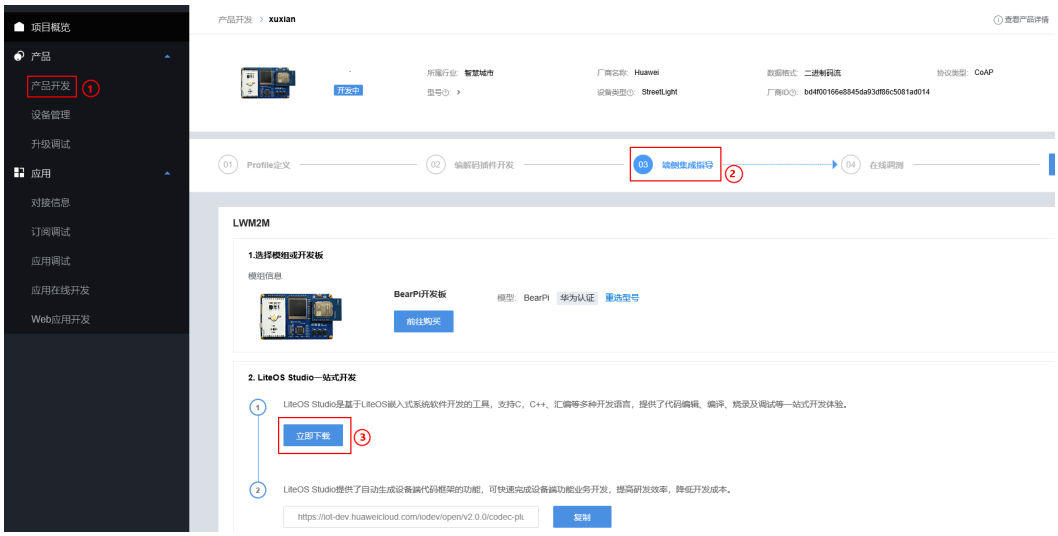
  b. 单击“添加属性”。在弹出窗口中配置属性的各项参数。如下图所示，添加“SignalPower”属性，单击“确定”。

c. 参照**步骤10.b**，依次添加“ECL”、“SNR”以及“CellID”属性。

所有服务都添加完成后，效果如下图所示。



步骤11 单击“端侧集成指导”，下载并安装IoT Studio。



步骤12 一键安装依赖包。依赖包下载地址：**下载链接**。

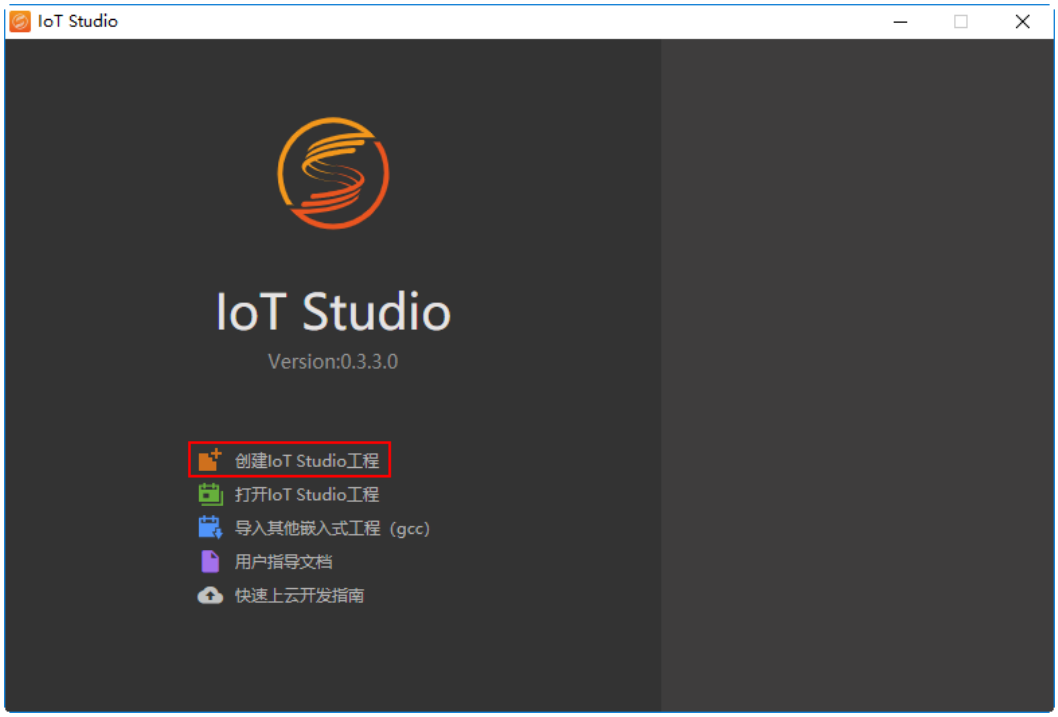
说明

依赖包均为开源软件，如需源码，请访问：**源码下载链接**。

有如下两种安装方式：

- （推荐方式）将依赖包解压到任意目录，运行InstallTools.bat，然后安装IDE。
- 将依赖包拷贝到IDE安装包的同级目录，然后将依赖包解压到当前目录，安装之后即可直接使用IDE。

步骤13 运行IoT Studio。单击“创建IoT Studio工程”。

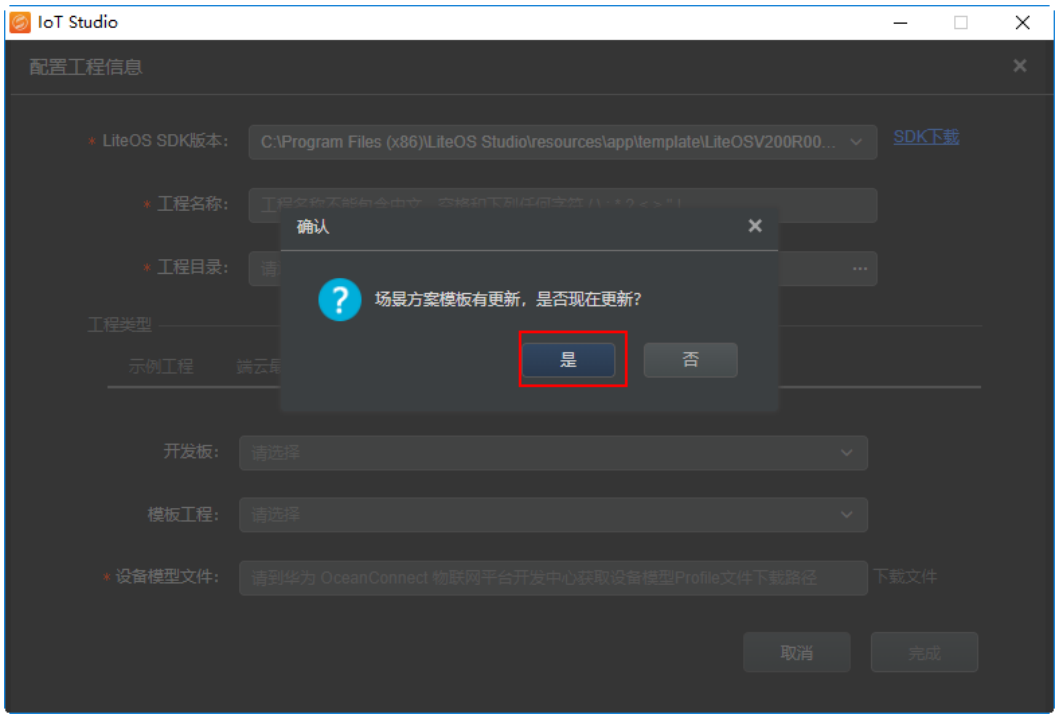


步骤14 自定义“工程名称”及选择“工程目录”。

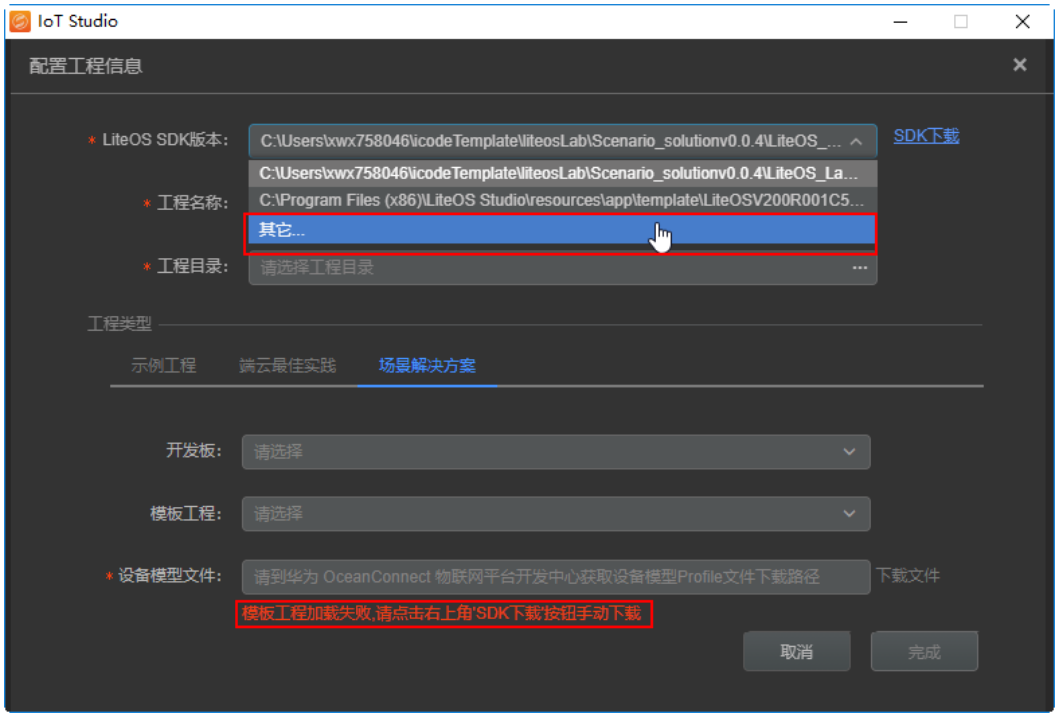
步骤15 单击“场景解决方案”页签。获取模板工程。

有如下两种方式获取模板工程：

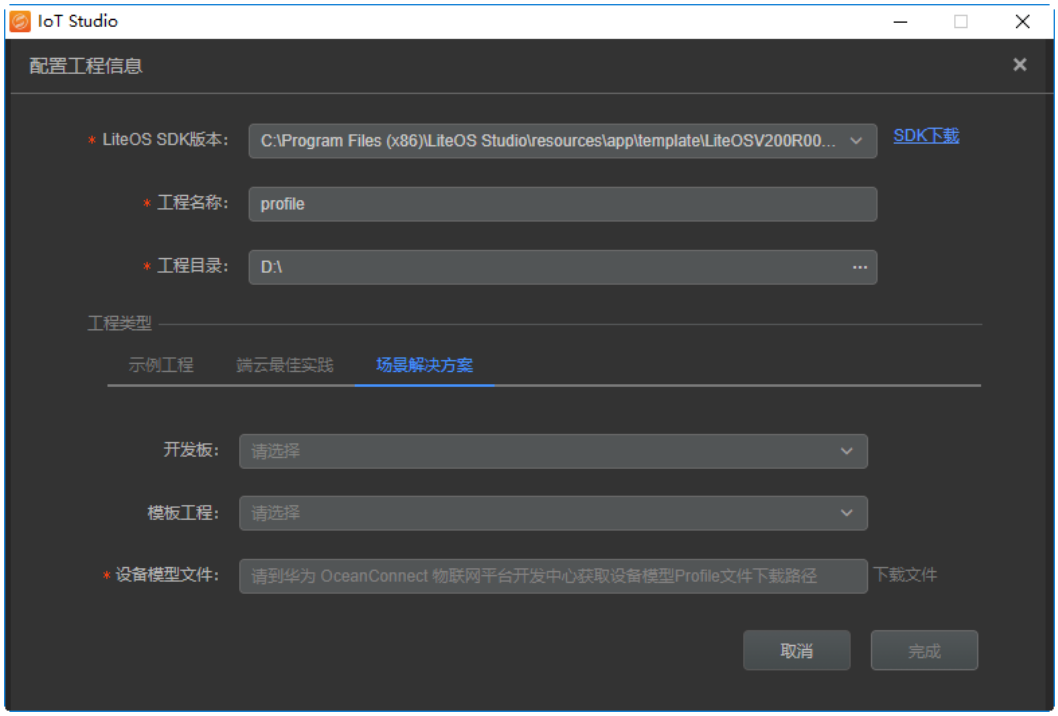
- 网络畅通时，界面会弹出如下提示，单击“是”更新模板。



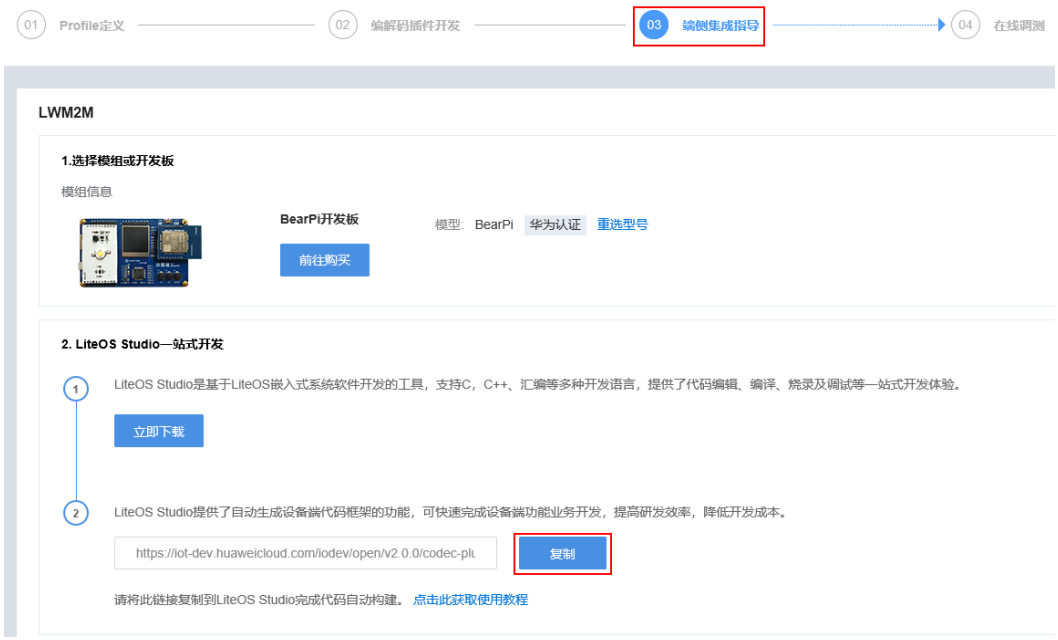
- 单击“SDK下载”手动下载模板，将下载的模板解压。单击“LiteOS SDK版本”后的下拉框，单击“其它”选项，手动选择下载的模板工程。



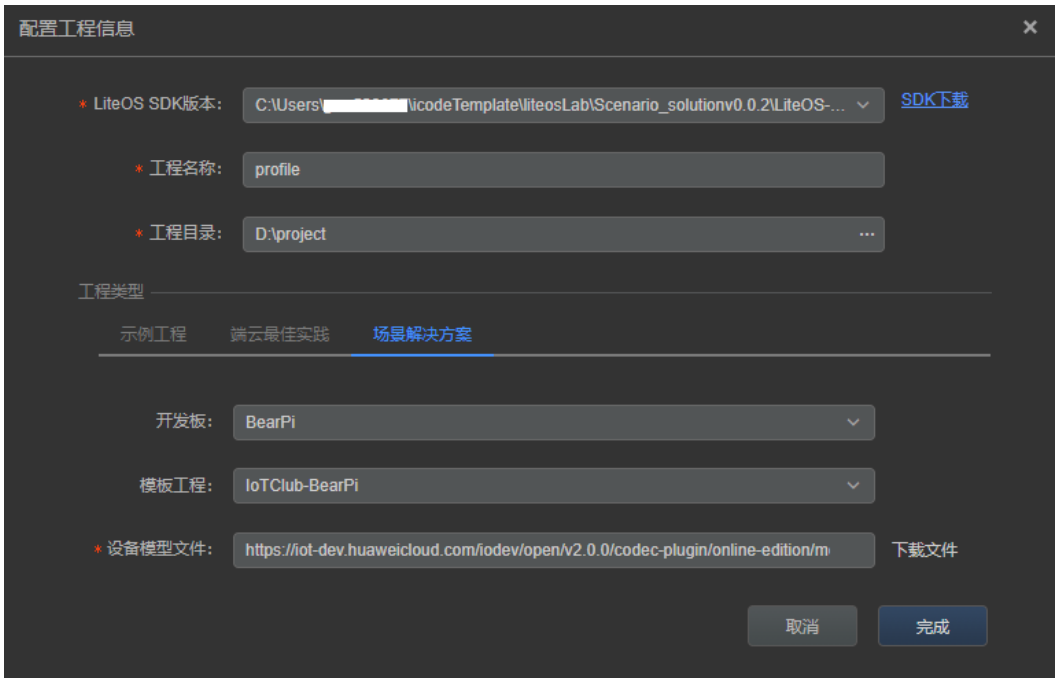
模板更新成功后，自动配置场景解决方案的“LiteOS SDK版本”、“开发板”及“模板工程”，如下图所示：



步骤16 在开发中心中，进入新创建的产品，选择“端侧集成指导”。如下图所示，单击“复制”，复制“设备模型文件”URL。



步骤17 在“设备模型文件”输入框中粘贴已复制的“设备模型文件”URL。单击“下载文件”。



步骤18 单击“完成”，将“设备模型文件”自动映射到端侧工程中，上云相关模块主要集中在产生的2个文件app.c、app.h中。

- app.c文件中已自动生成了上传命令和下发命令两个接口文件
- app.h文件主要描述映射生成的数据结构、硬件初始化接口

步骤19 在工程中打开app.c(“targets\IoTClub-BearPi\Apps”)文件。

1. 用户在未使用IoT Studio的profile自动映射功能前，app.c文件中的上传以及下发数据命令接口中内容为空，如下图所示，需要用户自己填写所有代码。

```
1
2   /*
3    * Name:app.c
4    *
5    *  Created on: 2019.5.27
6    *  Author: IoT Studio
7    *  Interface: sensor_controll_handler      sensor controll command from IoT cloud
8    *  Interface: report_data_handler   report data to IoT cloud command
9    */
10  #include "app.h"
11
12  int sensor_controll_handler()
13  {
14
15
16
17  }
18
19  int report_data_handler()
20  {
21
22
23
24  }
```

2. 下发命令接口中已包含了解析数据的代码，如下图所示，用户只需要在红框中的区域1内填写下发命令的应用逻辑处理代码即可。


```
int sensor_controll_handler()
{
// Define data report struct, all members come from devicemodel file
#if 1
    tag_app_Response_Set_Led Response_Set_Led;
    tag_app_Set_Led Set_Led;
#endif

    if (semp_pend(s_rcv_sync, 0x7fffffff))
    {
        uint8_t msgid = s_rcv_buffer[0];
        switch (msgid)
        {
#if 1
        case cn_app_Set_Led:
            memcpy(&Set_Led, s_rcv_buffer, sizeof(tag_app_Set_Led));
            /********** code area for cmd from IoT cloud  **********/
                        1
            /********** code area end  **********/
            break;
        default:
            break;
#endif
        }
    }
}
```

3. 上传命令接口中已包含了解析数据的代码，并且默认映射生成了虚拟数据，保证上传通道正常。如下图所示，用户只需在红色框中的区域2填写真实传感数据上传命令处理逻辑的代码即可。


```
int report_data_handler()
{
#if 1
    tag_app_Report_Connectivity Report_Connectivity;
    tag_app_Report_Toggle Report_Toggle;
    tag_app_Report_Sensor Report_Sensor;
#endif

/********** code area for report data to IoT cloud  **********/
                    2
/********** code area end  **********/

// virtual sensor data example
#if 1

    Report_Connectivity.messageId = cn_app_Report_Connectivity;
    Report_Connectivity.SignalPower = htons(2);
    Report_Connectivity.ECL = htons(2);
    Report_Connectivity.SNR = htons(2);
    Report_Connectivity.CellID = htonl(4);
    oc_lwm2m_report((uint8_t *)&Report_Connectivity, sizeof(Report_Connectivity), 1000);

    Report_Toggle.messageId = cn_app_Report_Toggle;
    Report_Toggle.toggle = htons(2);
    oc_lwm2m_report((uint8_t *)&Report_Toggle, sizeof(Report_Toggle), 1000);

    Report_Sensor.messageId = cn_app_Report_Sensor;
    Report_Sensor.data = htons(2);
    oc_lwm2m_report((uint8_t *)&Report_Sensor, sizeof(Report_Sensor), 1000);
#endif
}
```

步骤20 单击“Ctrl+S”快捷键保存修改的“app.c”文件。

步骤21 在工具栏中单击  ，对当前工程进行编译。编译成功后，在控制台面板中显示“编译成功”。如下图所示。

```
控制台

arm-none-eabi-size  build/Huawei_LiteOS.elf
text  data  bss  dec  hex  filename  78264  512  8968  87744  156c0  build/Huawei_LiteOS.elf
arm-none-eabi-objcopy  -O  ihex  build/Huawei_LiteOS.elf  build/Huawei_LiteOS.hex
arm-none-eabi-objcopy  -O  binary  -S  build/Huawei_LiteOS.elf  build/Huawei_LiteOS.bin
0  Error(s),  12  Warning(s).
[2019-07-17  15:02:20]  编译成功。  编译耗时：8459ms
```

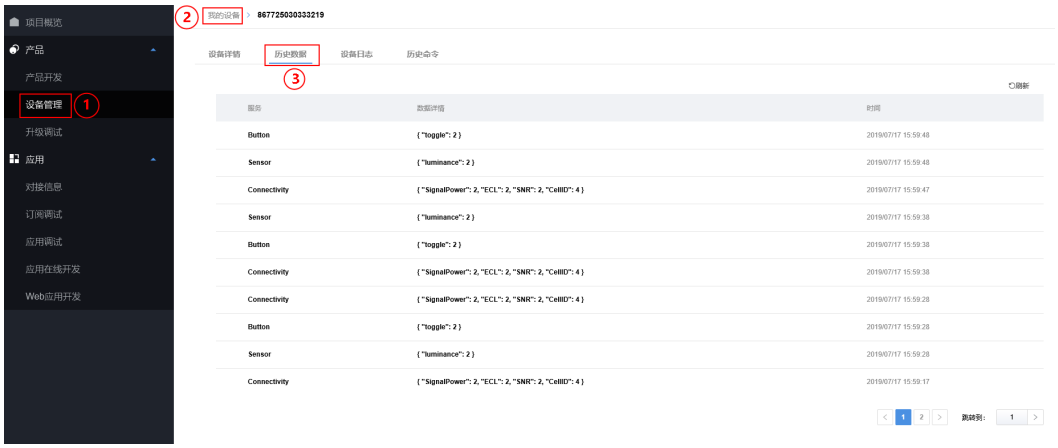
步骤22 使用数据线将开发板与电脑连接。

步骤23 单击工具栏中的  ，将已经编译的程序烧录至开发板。

----结束

操作结果

1. 如下图所示，云端将会显示数据上传的结果。



2. 若云端无上传数据，请参考**通信模组检测工具使用指导**以及**常见问题**定位分析端云通信故障问题。

任务示例

真实设备数据参考实现：本节将在上述虚拟数据上云工程的基础上，以小熊派开发板真实光强数据采集DEMO 为例，介绍**初始化光强检测传感器**、**处理真实LED灯开关控制**等场景如何修改代码并将数据上传到云端。

步骤1 命令下发处理：

1. 在***app.c***文件中查找***sensor_controll_handler ()*** 函数
2. 在 ***/********** code area for cmd from IoT cloud **********/*** 注释下添加下面段代码

```
//云端下发命令处理
    printf("Set_Led:msgid:%d status:%s", Set_Led.messageId, Set_Led.led);
    char* match = Set_Led.led;
//开灯
    if(strstr(match,"ON")!=NULL) {
        HAL_GPIO_WritePin(Light_GPIO_Port, Light_Pin, GPIO_PIN_SET);
    }
//关灯
    if (strstr(match,"OFF")!=NULL) {
         HAL_GPIO_WritePin(Light_GPIO_Port, Light_Pin, GPIO_PIN_RESET);
    }
```


添加完代码后界面如下图所示。

```
    if (semp_pend(s_rcv_sync, 0x7fffffff))
    {
        uint8_t msgid = s_rcv_buffer[0];
        switch (msgid)
        {
#if 1
        case cn_app_Set_Led:
            memcpy(&Set_Led, s_rcv_buffer, sizeof(tag_app_Set_Led));
            /********** code area for cmd from IoT cloud  **********/

            //云端下发命令处理
            printf("Set_Led:msgid:%d status:%s", Set_Led.messageId, Set_Led.led);
            char* match = Set_Led.led;
            //开灯
            if(strstr(match,"ON")!=NULL) {
                HAL_GPIO_WritePin(Light_GPIO_Port, Light_Pin, GPIO_PIN_SET);
            }
            //关灯
            if (strstr(match,"OFF")!=NULL) {
                HAL_GPIO_WritePin(Light_GPIO_Port, Light_Pin, GPIO_PIN_RESET);
            }

            /********** code area end  **********/
            break;
        default:
            break;
#endif
        }
    }
```

步骤2 数据上报处理：

1. 在***app.h***头文件中增加***#include <bh1750.h>***，并将***app_init()*** 函数代码，整体替换为下面段代码。

```
static VOID app_data_collection_task(VOID)
{
    UINT32 uwRet = LOS_OK;
    Init_BH1750();
    while (1)
    {
        Lux = (int)Convert_BH1750();
        LCD_ShowString(10, 200, 200, 16, 16, "BH1750 Value is:");
        LCD_ShowNum(140, 200, Lux, 5, 16);

        uwRet = LOS_TaskDelay(2000);
        if (uwRet != LOS_OK)
            return;
    }
}

int app_init()
{
    semp_create(&s_rcv_sync,1,0);
    task_create("app_report", (fnTaskEntry)app_report_task_entry, 0x1000, NULL, NULL, 2);
    task_create("app_command", (fnTaskEntry)app_cmd_task_entry, 0x1000, NULL, NULL, 3);
    task_create("data_collection_task", (fnTaskEntry)app_data_collection_task, 0x1000, NULL,
NULL, 3);
    return 0;
}
```

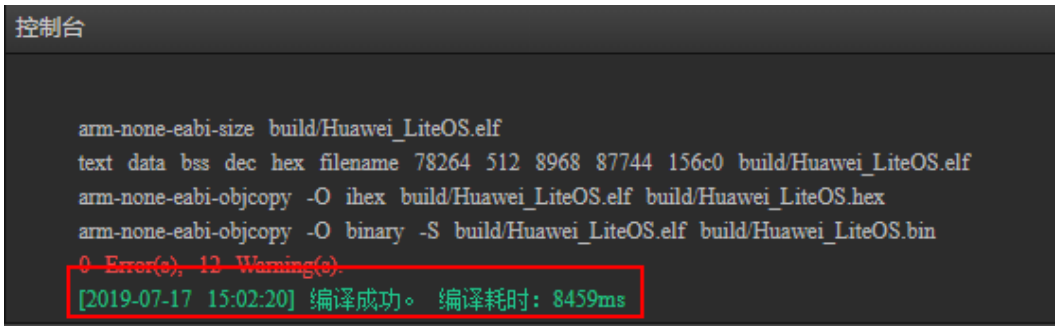
2. 在***app.c***文件中查找***report_data_handler ()*** 函数，在 ***/********** code area for report data to IoT cloud **********/*** 注释下添加下面段代码

```
Report_Sensor.messageId = cn_app_Report_Sensor;
Report_Sensor.data = htons(2);
oc_lwm2m_report((uint8_t *)&Report_Sensor, sizeof(Report_Sensor), 1000);
```

添加完代码后界面如下图所示。

```
int report_data_handler()
{
#if 1
    tag_app_Report_Connectivity Report_Connectivity;
    tag_app_Report_Toggle Report_Toggle;
    tag_app_Report_Sensor Report_Sensor;
#endif

/********** code area for report data to IoT cloud  **********/
//真实光强采集数据
Report_Sensor.messageId = cn_app_Report_Sensor;
Report_Sensor.data = htons(Lux);
oc_lwm2m_report((uint8_t *)&Report_Sensor, sizeof(Report_Sensor), 1000);
/********** code area end  **********/

// virtual sensor data example
#if 1

    Report_Connectivity.messageId = cn_app_Report_Connectivity;
    Report_Connectivity.SignalPower = htons(2);
    Report_Connectivity.ECL = htons(2);
    Report_Connectivity.SNR = htons(2);
    Report_Connectivity.CellID = htonl(4);
    Report_Connectivity.testarray[0] = 8;
    Report_Connectivity.testarray[1] = 8;
    Report_Connectivity.testarray[2] = 8;
    oc_lwm2m_report((uint8_t *)&Report_Connectivity, sizeof(Report_Connectivity), 1000);

    Report_Toggle.messageId = cn_app_Report_Toggle;
    Report_Toggle.toggle = htons(2);
    oc_lwm2m_report((uint8_t *)&Report_Toggle, sizeof(Report_Toggle), 1000);

    Report_Sensor.messageId = cn_app_Report_Sensor;
    Report_Sensor.data = htons(2);
    oc_lwm2m_report((uint8_t *)&Report_Sensor, sizeof(Report_Sensor), 1000);
#endif
```

步骤3 在工具栏中单击  ，对当前工程进行编译。编译成功后，在控制台面板中显示“编译成功”。如下图所示。



步骤4 单击工具栏中的  ，将已经编译的程序烧录至开发板。

----结束

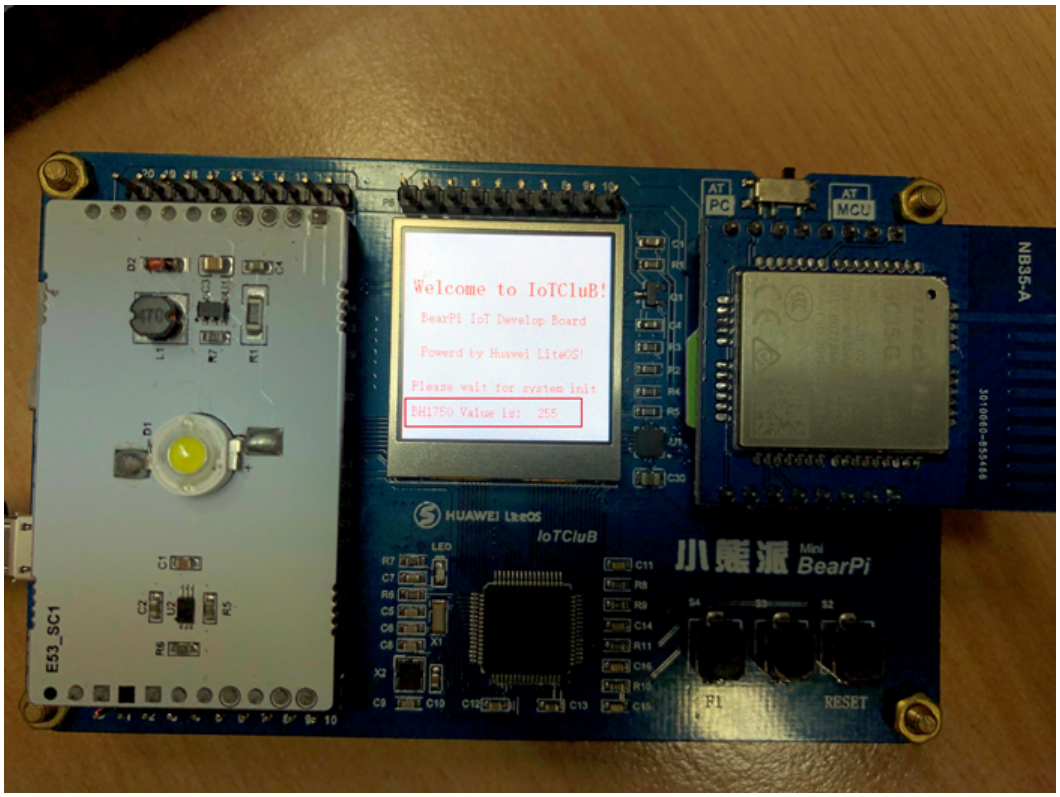
操作结果

1. 云端查看上报的光强数据。

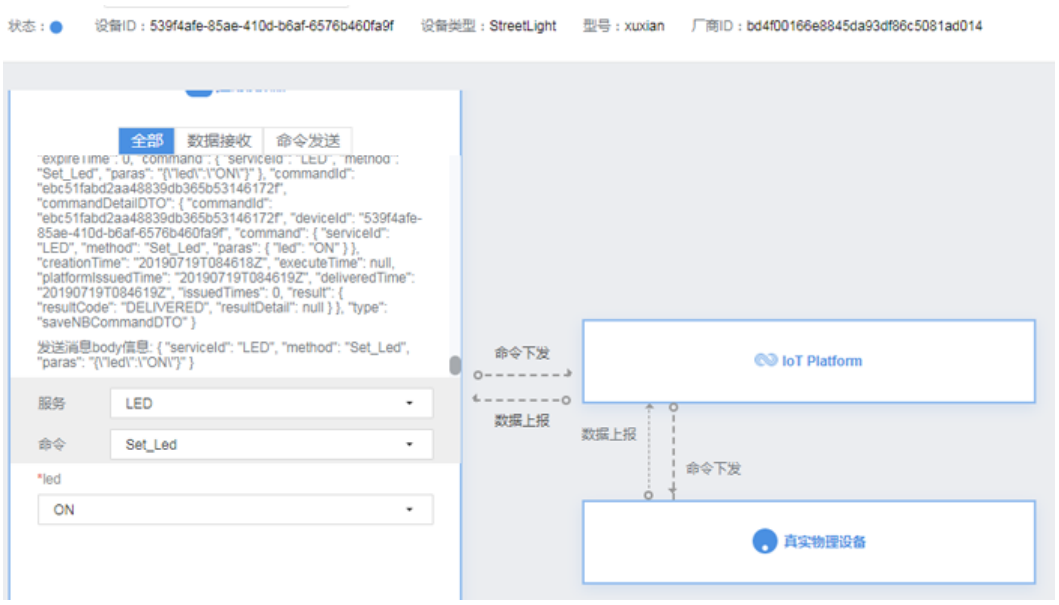
设备详情　历史数据　设备日志　历史命令

服务	数据详情	时间
Sensor	{ "luminance": 284 }	2019/07/19 16:42:53 GTM+08:00
Sensor	{ "luminance": 2 }	2019/07/19 16:42:53 GTM+08:00
Connectivity	{ "SignalPower": 2, "ECL": 2, "SNR": 2, "CellID": 4 }	2019/07/19 16:42:42 GTM+08:00
Sensor	{ "luminance": 2 }	2019/07/19 16:42:42 GTM+08:00
Sensor	{ "luminance": 243 }	2019/07/19 16:42:42 GTM+08:00
Button	{ "toggle": 2 }	2019/07/19 16:42:42 GTM+08:00

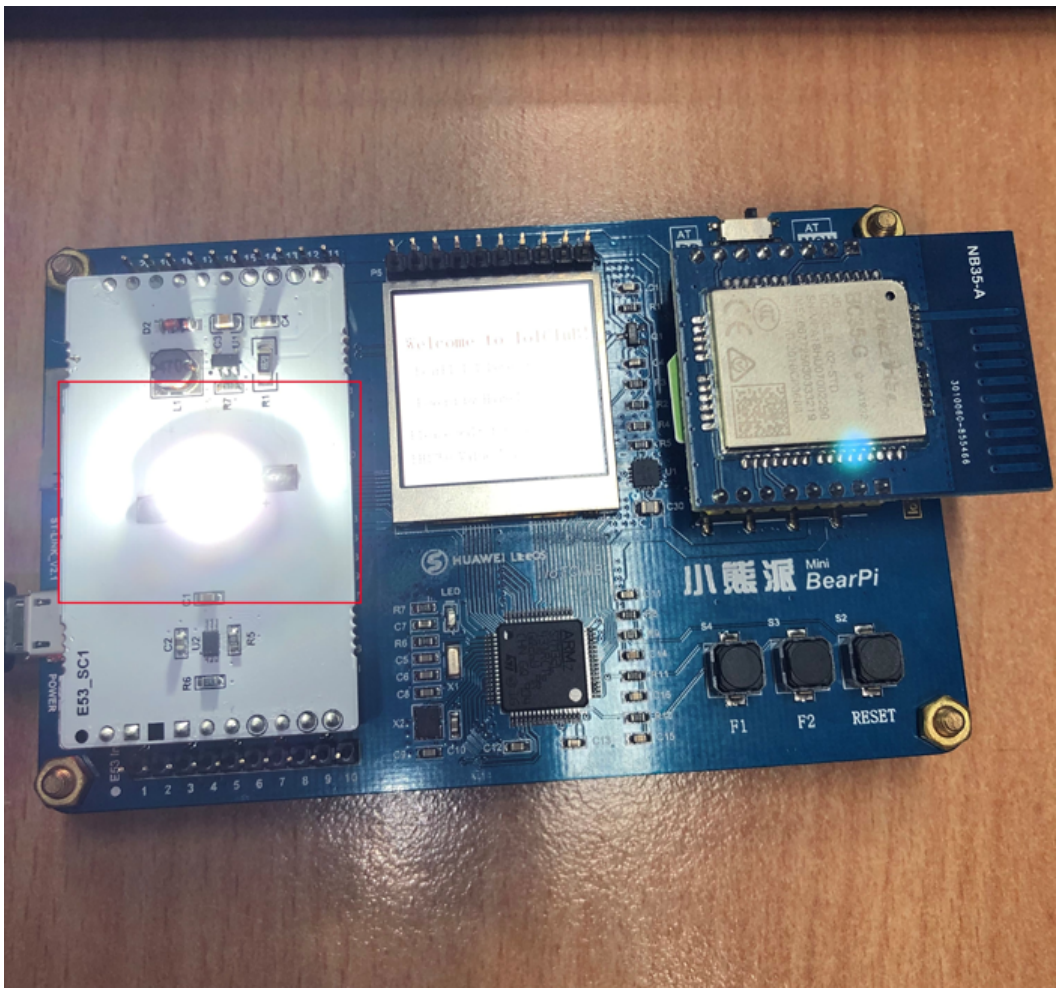
2. 设备端查看实时光强数据。



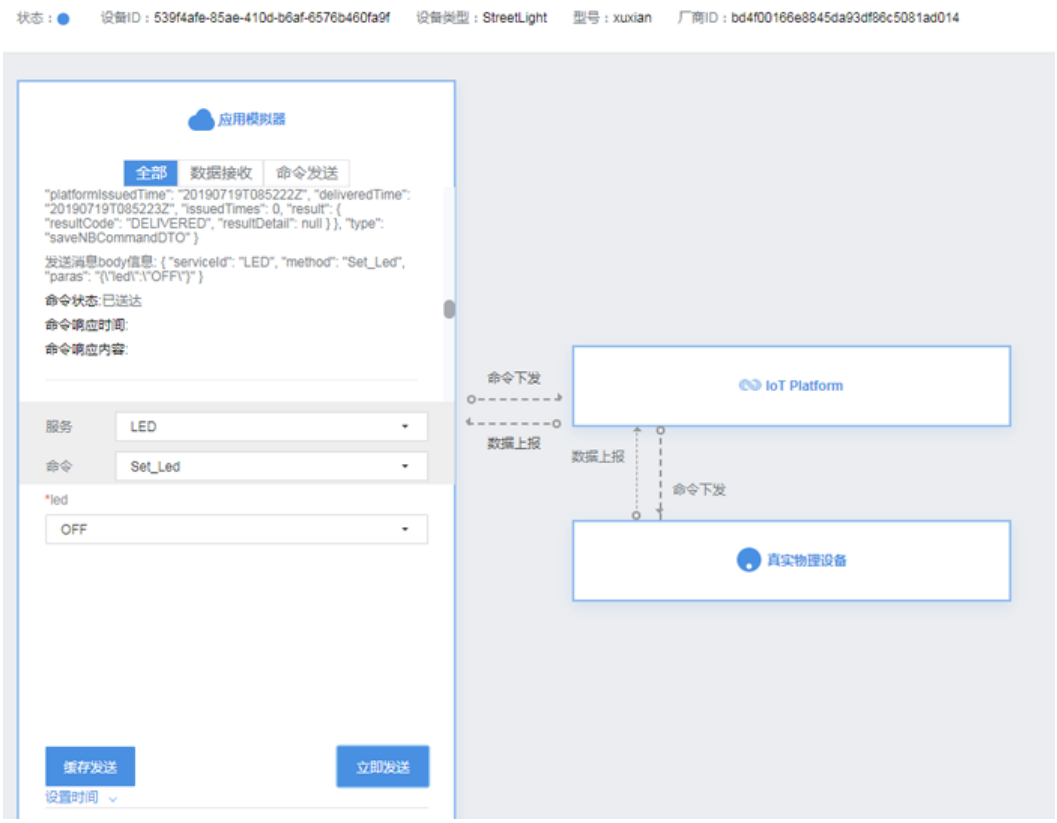
3. 云侧下发开灯命令。



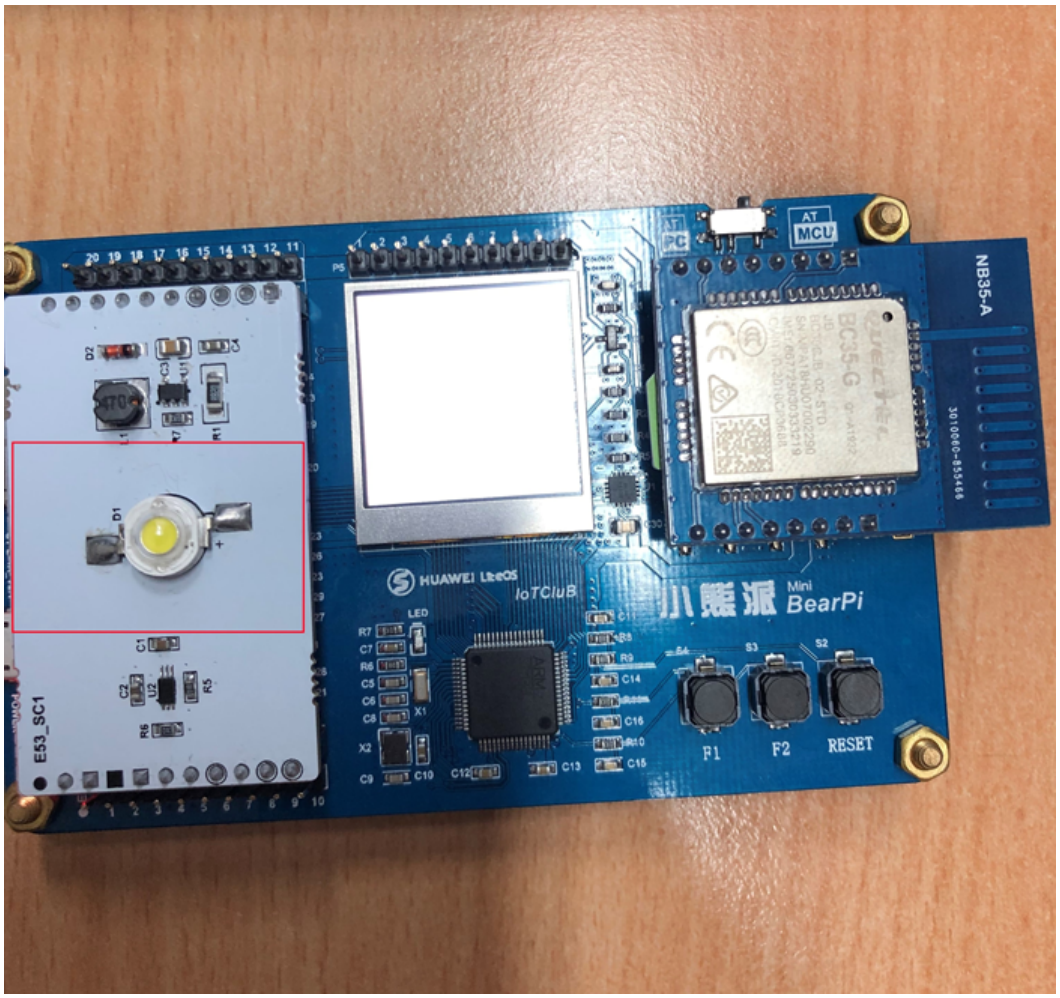
4. 端侧响应云端命令开灯。



5. 云侧下发关灯命令。



6. 端侧响应关灯命令。



开发框架介绍

- Huawei LiteOS简介：

  Huawei LiteOS是华为面向物联网领域开发的一个基于实时内核的轻量级操作系统。本项目属于华为物联网操作系统Huawei LiteOS源码，现有基础内核支持任务管理、内存管理、时间管理、通信机制、中断管理、队列管理、事件管理、定时器等操作系统基础组件，更好地支持低功耗场景，支持tickless机制，支持定时器对齐。

  Huawei LiteOS提供端云协同能力，集成了LwM2M、CoAP、mbedtls、LwIP全套IoT互联协议栈，且在LwM2M的基础上，提供了AgentTiny模块，用户只需关注自身的应用，而不必关注LwM2M实现细节，直接使用AgentTiny封装的接口即可简单快速实现与云平台安全可靠的连接。

  Huawei LiteOS自开源社区发布以来，围绕NB-IoT物联网市场从技术、生态、解决方案、商用支持等多维度使能合作伙伴，构建开源的物联网生态，推出一批开源开发套件和行业解决方案，帮助众多行业客户快速的推出物联网终端和服务，客户涵盖抄表、停车、路灯、环保、共享单车、物流等众多行业，为开发者提供“一站式”完整软件平台，有效降低开发门槛、缩短开发周期。

- LiteOS_Lab仓库与LiteOS仓库的关系：

  LiteOS_Lab的Lab是实验室英文单词的简写，用于一些新的特性快速开发，概念验证，第三方代码合入。

- LiteOS_Lab roadmap
  - 读写锁（初步完成，原定时间点2018.12.31）
  - 驱动框架（初步完成，原定时间点2018.12.31）
  - MPU支持（初步完成，原定时间点2018.12.31）
  - 静态创建任务（初步完成，原定时间点2018.12.31）
  - 内存管理改进（初步完成，部分target依然使用旧版本，原定时间点2019.1.31）
  - SVC调用（初步完成，IAR未支持）
  - cortex-m arch代码去冗余，M3，M4，M7公用代码（初步完成）
  - shell框架以及一组简单命令（初步完成）
  - AT命令解析框架（规划中）

  说明

  更多代码框架及目录结构的详细说明，请访问：**https://github.com/LiteOS/LiteOS_Lab**进行详细了解。

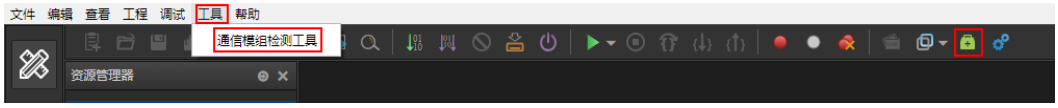
  如有疑问或想与我们进行交流，请访问：**https://bbs.huaweicloud.com/forum/forum-727-1.html**联系我们。

4.1.3.2 通信模组检测工具使用指导

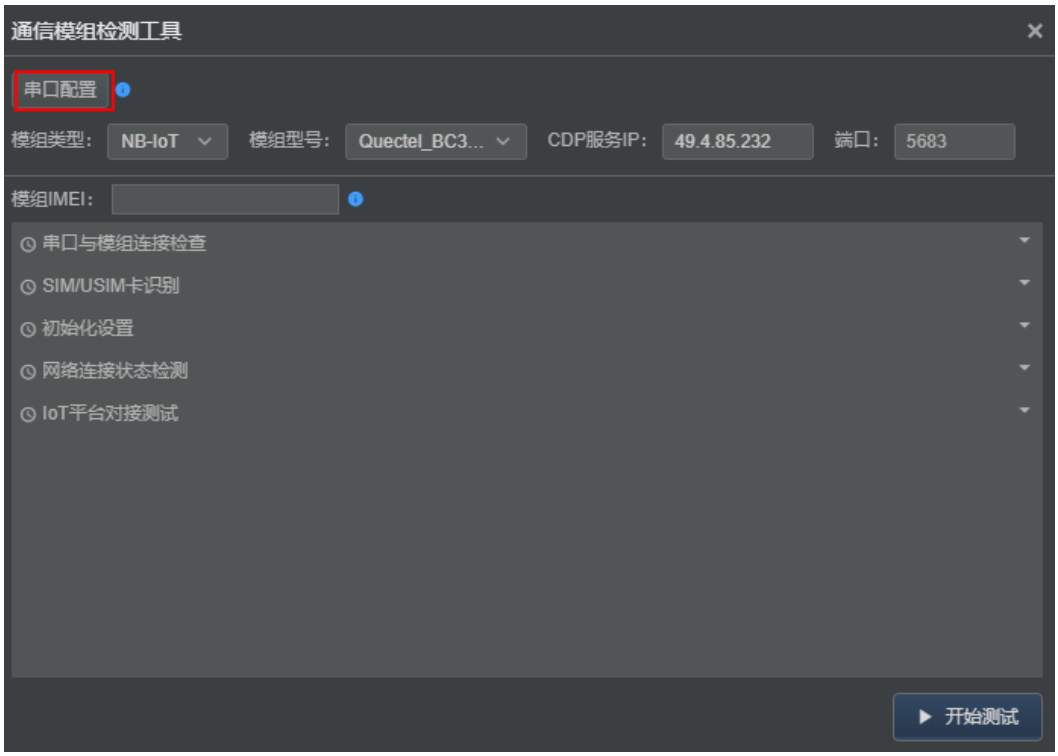
IoT Studio在与OC云端连通使用时，可使用通信模组检测工具快速定位Quectel_BC35-G&BC28&BC95模组与云端连通性问题，提高开发效率。本节将以小熊派开发板为例介绍如何使用通信模组检测工具定位问题。

操作步骤

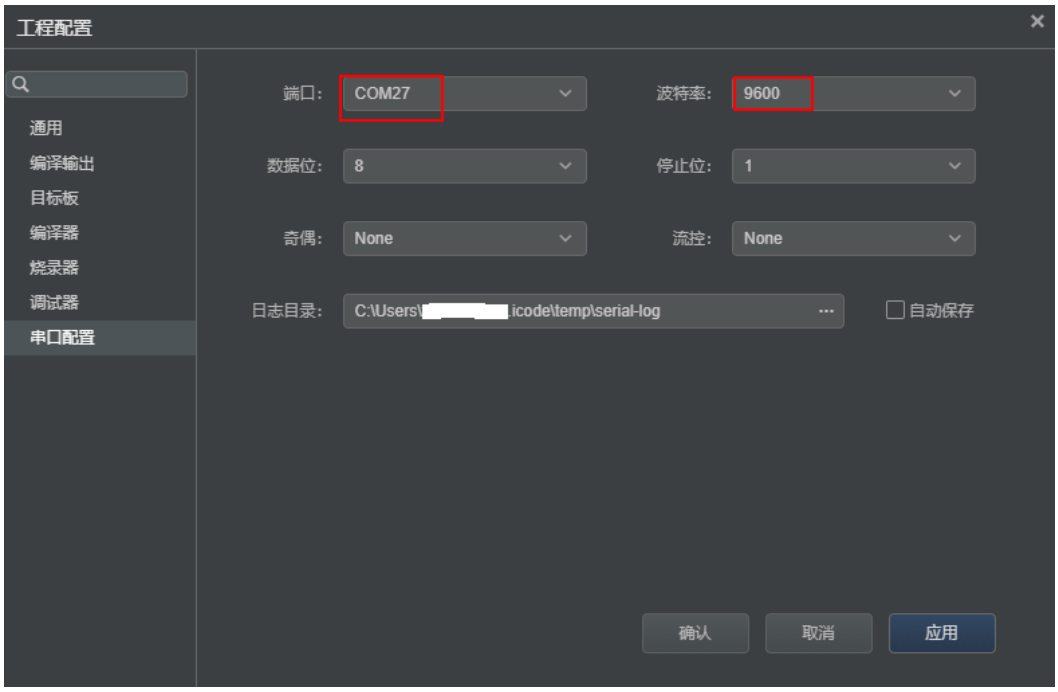
步骤1 单击菜单栏中的“工具>通信模组检测工具”或直接单击工具栏中的图标  ，打开通信模组检测工具。



步骤2 单击“串口配置”。



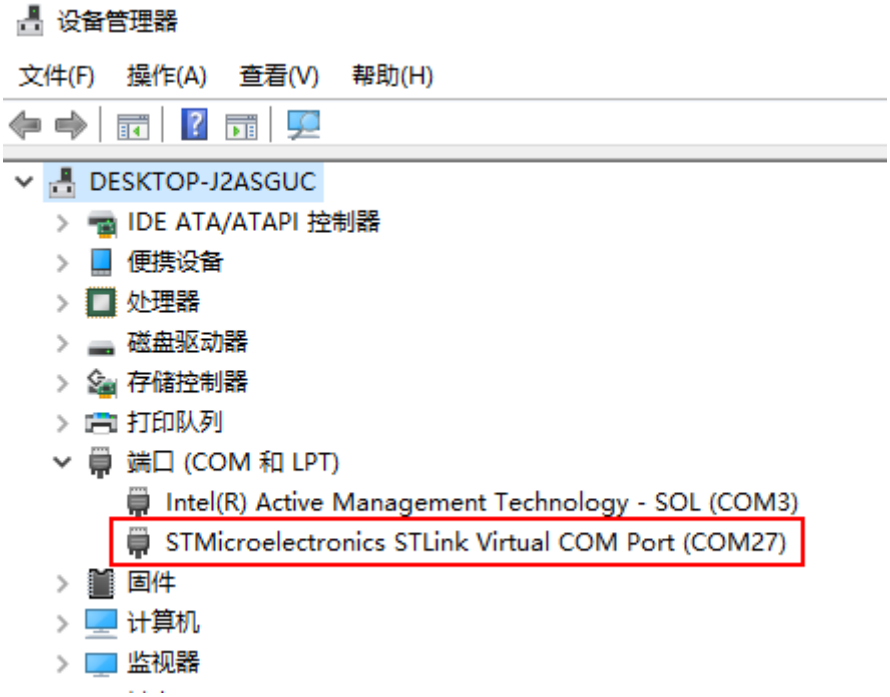
步骤3 在“串口配置”界面将“波特率”配置为“9600”。在“端口”下拉框中选择与开发板实际连接的端口号。获取端口号具体方法请参见下方说明。



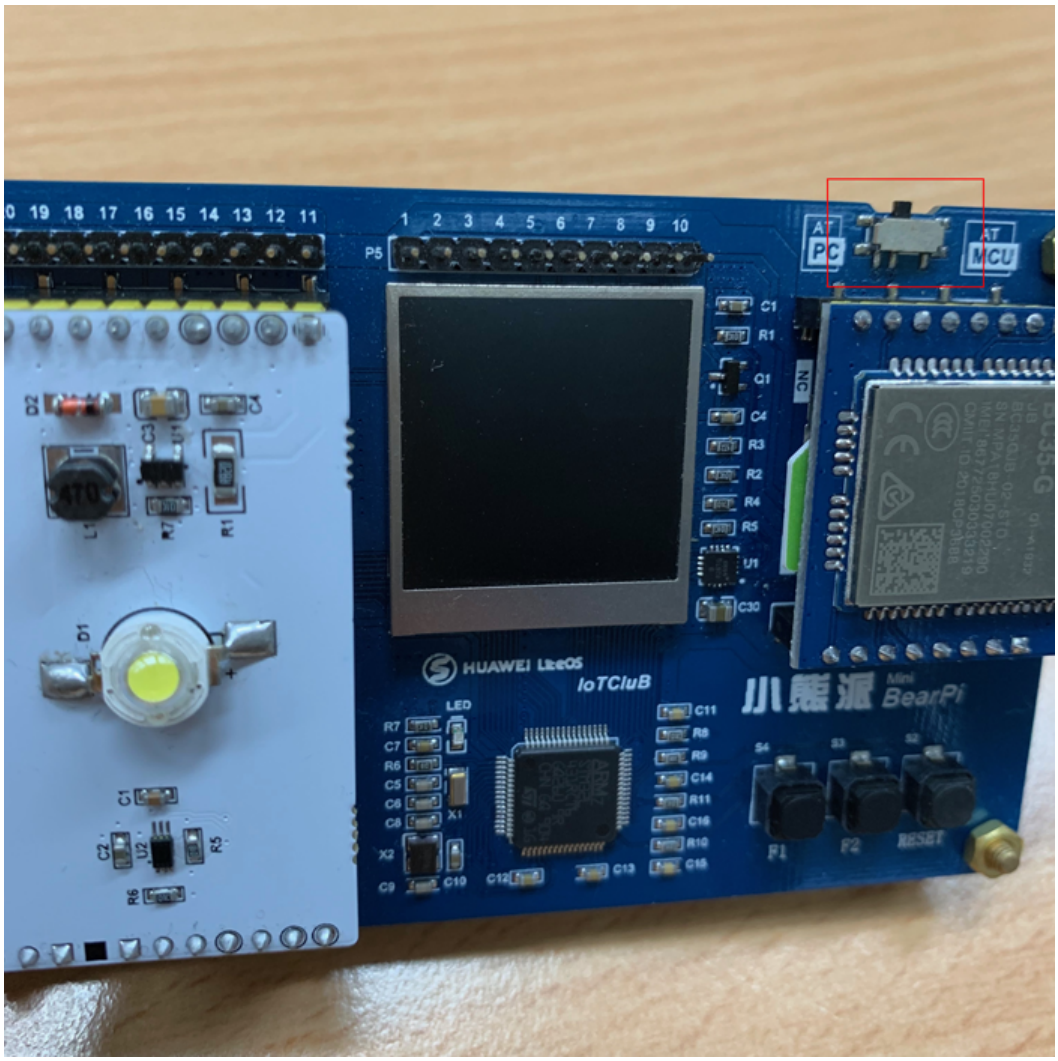
说明

获取端口号方式如下：

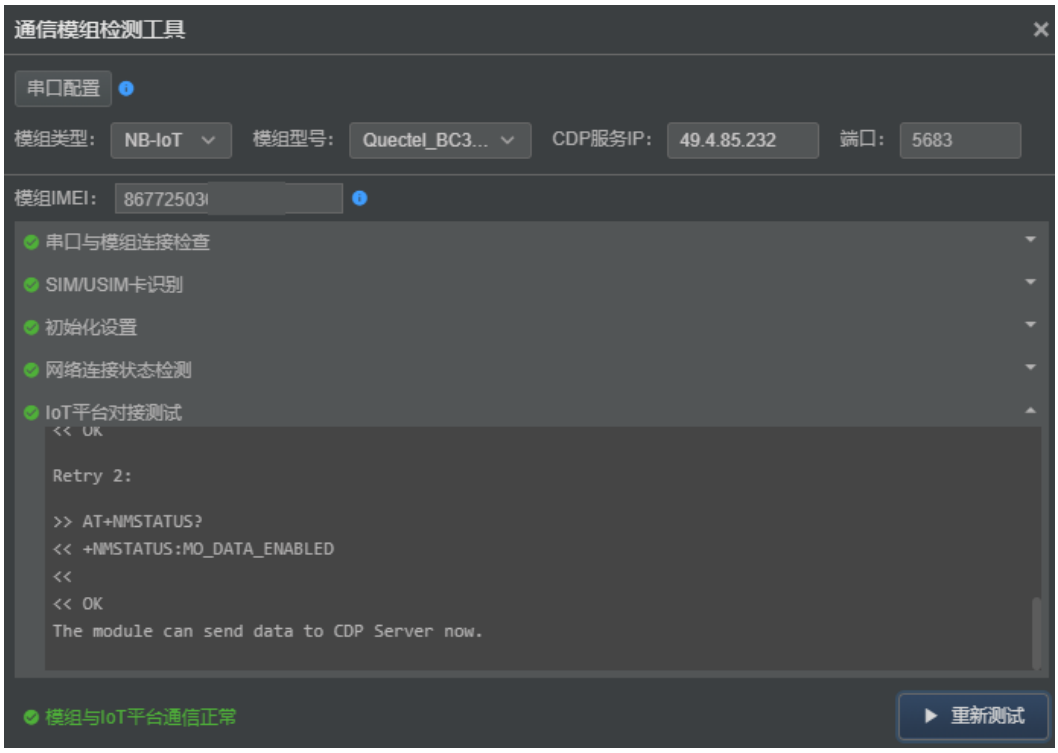
1. 打开“控制面板\硬件和声音\设备管理器”。
2. 在“端口”子菜单下，找到连接设备的串口号，如下图所示：



步骤4 将与PC连接的通信模组上的开关拨至PC侧。



步骤5 单击“开始测试”。若通信正常，测试结果如下图所示。

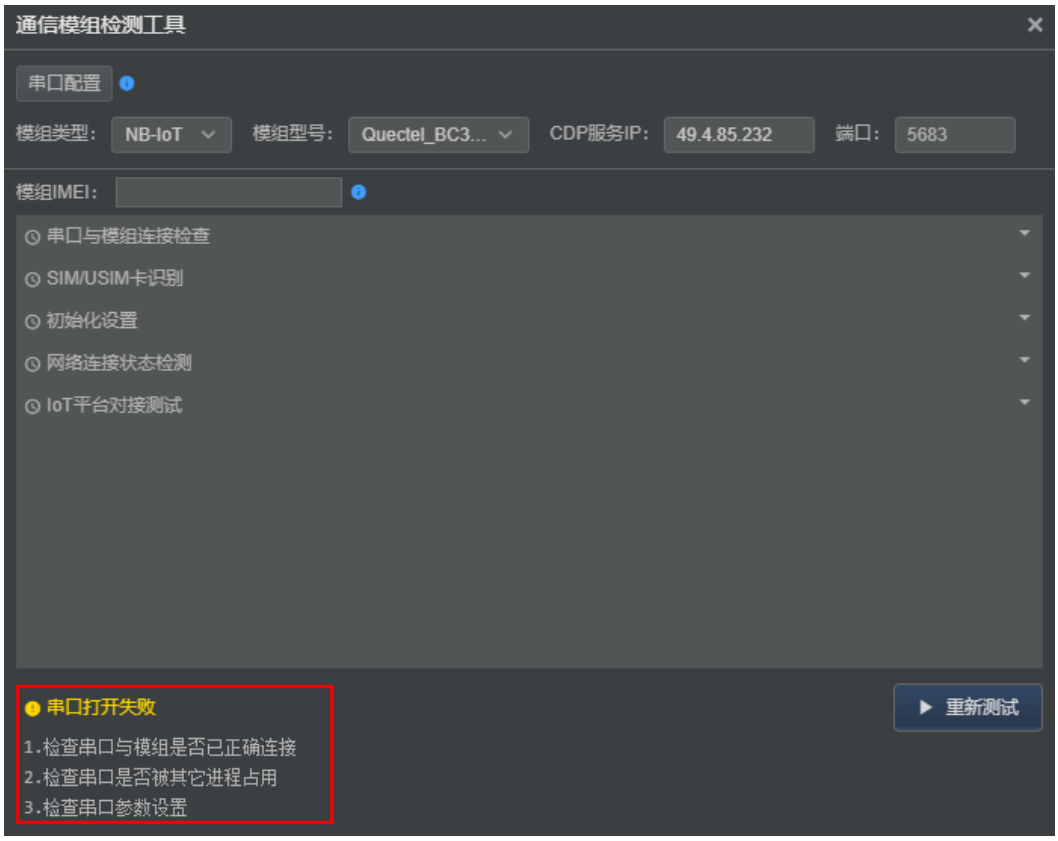


----结束

4.1.3.3 常见问题

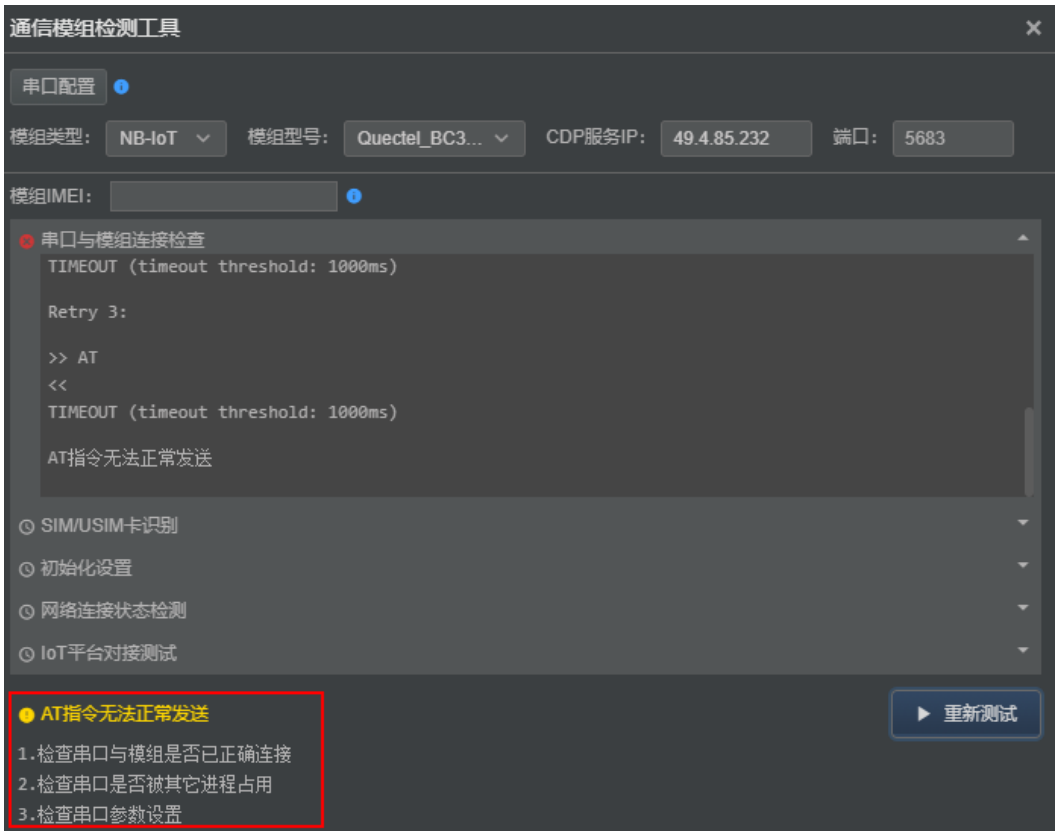
4.1.3.3.1 场景一：串口打开失败

检测后界面弹出如下提示，请检查串口配置是否正确，端口号是否为实际端口号，波特率是否设置为“9600”。



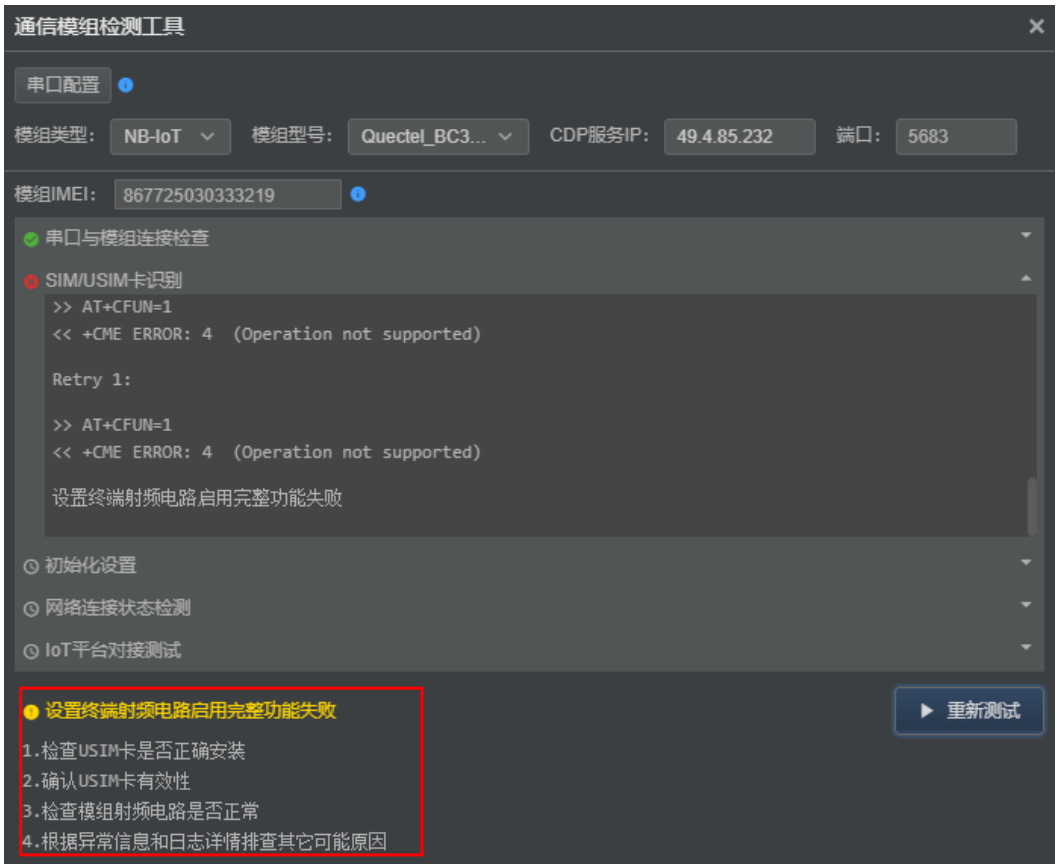
4.1.3.3.2 场景二：模组连接异常/模组损坏

检测后界面弹出如下提示，请检查开发板上的开关是否已拨至PC侧、模组是否损坏、以及模组是否正确的插入卡槽。



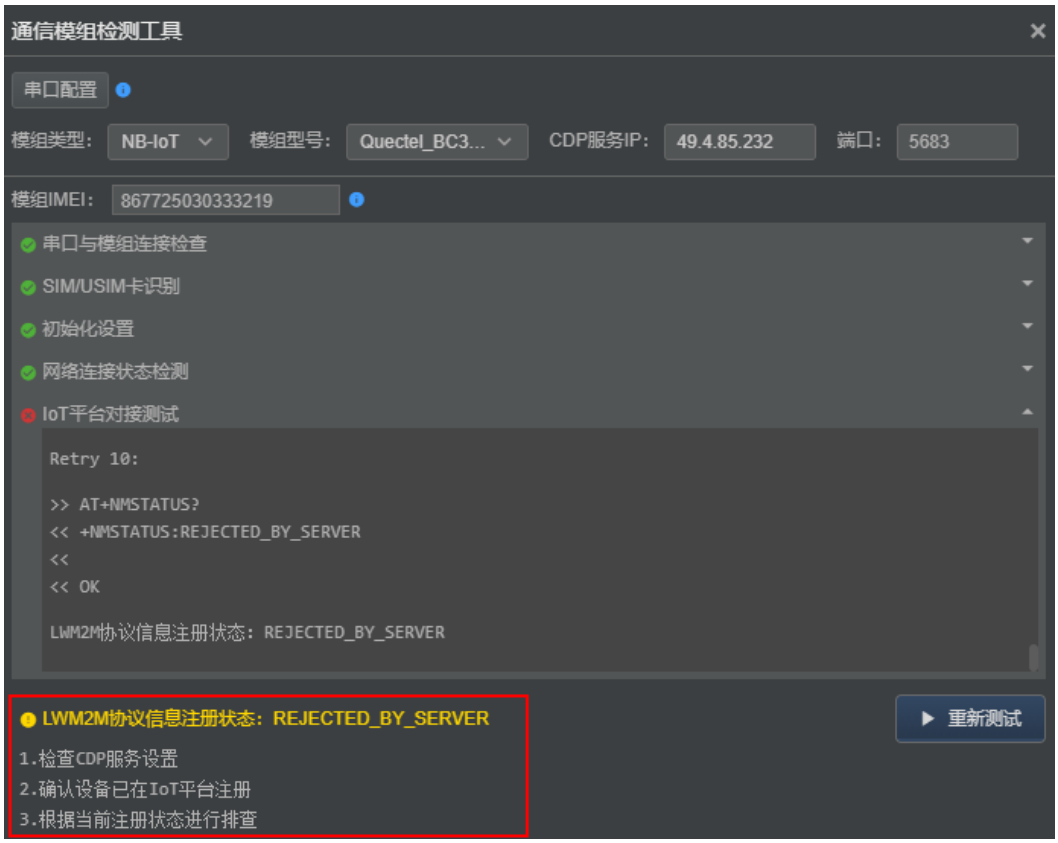
4.1.3.3.3 场景三：SIM 未正确插入卡槽

检测后界面弹出如下提示，请检查开发板上的SIM是否正反面或者方向插错，以及SIM卡的有效性。



4.1.3.3.4 场景四：模组未在云端注册

检测后界面弹出如下提示，请在云端注册该模组。



通过上述全流程操作步骤，我们能够端到端进行端侧的开发和调测，确保数据的正确上传。

4.1.4 附录 LwM2M 协议介绍

4.1.4.1 LwM2M 协议是什么

LwM2M（Lightweight M2M，轻量级M2M），由开发移动联盟（OMA）提出，是一种轻量级的、标准通用的物联网设备管理协议，可用于快速部署客户端/服务器模式的物联网业务。

LwM2M为物联网设备的管理和应用建立了一套标准，它提供了轻便小巧的安全通信接口及高效的数据模型，以实现M2M设备管理和服务支持。

4.1.4.2 LwM2M 协议特性

LwM2M协议主要特性包括：

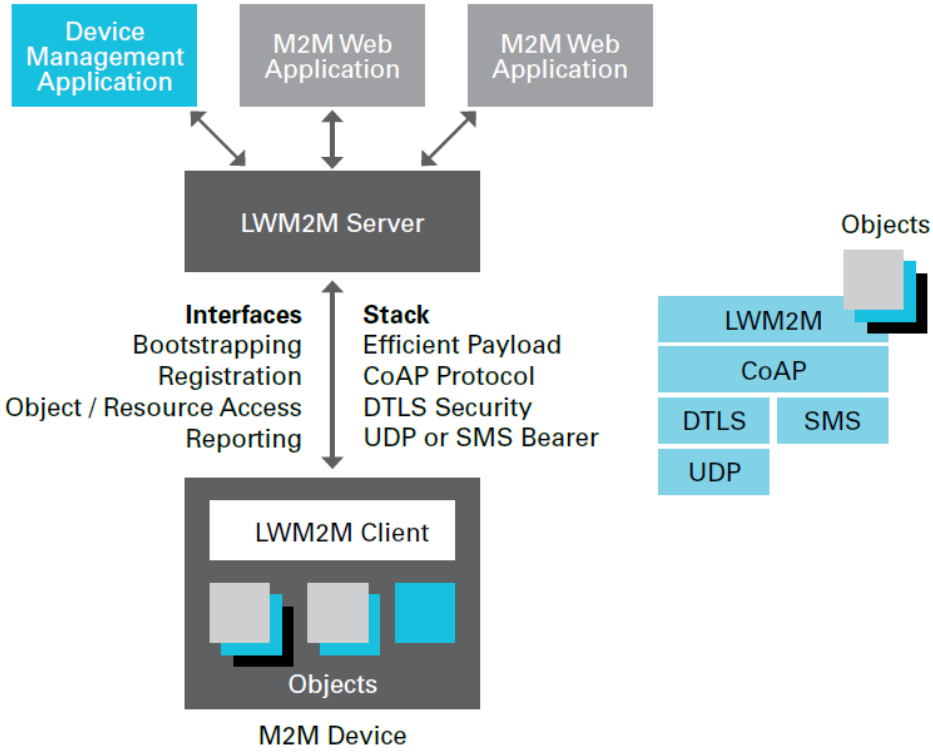
- 基于资源模型的简单对象
- 资源操作：创建/检索/更新/删除/属性配置
- 资源的观察/通知
- 支持的数据格式：TLV/JSON/Plain Text/Opaque
- 传输层协议：UDP/SMS
- 安全协议：DTLS
- NAT/防火墙应对方案: Queue模式

- 支持多LwM2M Server
- 基本的M2M功能：LwM2M Server，访问控制，设备，网络连接监测，固件更新，位置和定位服务，统计

4.1.4.3 LwM2M 体系架构

LwM2M体系架构如图所示：

图 4-7 LwM2M 体系架构



4.1.4.4 LwM2M 对象定义

对象概念

对象是逻辑上用于特定目的的一组资源的集合。例如固件更新对象，它就包含了用于固件更新目的的所有资源，例如固件包、固件URL、执行更新、更新结果等。

使用对象的功能之前，必须对该对象进行实例化，对象可以有多个对象实例，对象实例的编号从0开始递增。

OMA定义了一些标准对象，LwM2M协议为这些对象及其资源已经定义了固定的ID。例如：固件更新对象的对象ID为5，该对象内部有8个资源，资源ID分别为0~7，其中“固件包名字”这个资源的ID为6。因此，URI 5/0/6表示：固件更新对象第0个实例的固件包名字这个资源。

对象定义的格式

Name	Object ID	Instances	Mandatory	Object URN
Object Name	16-bit Unsigned Integer	Multiple/ Single	Mandatory/ Optional	urn:oma:LwM2M: {oma,ext,x}:{Object ID}

OMA 定义的标准对象

OMA的LwM2M规范中定义了7个标准对象：

Object	object id	description
LwM2M Security	0	LwM2M (bootstrap) server的URI，payload的安全模式，一些算法/密钥，server的短ID等信息。
LwM2M Server	1	server的短ID，注册的生命周期，observe的最小/最大周期， 绑定模型等。
Access Control	2	每个Object的访问控制权限。
Device	3	设备的制造商，型号，序列号，电量，内存等信息。
Connectivity Monitoring	4	网络制式，链路质量，IP地址等信息。
Firmware	5	固件包，包的URI，状态，更新结果等。
Location	6	经纬度，海拔，时间戳等。
Connectivity Statistics	7	收集期间的收发数据量，包大小等信息。

LiteOS SDK端云互通组件配合Huawei Ocean Connect物联网开发平台能力，还支持的19号LwM2M APPDATA对象：

Object	object id	description
LwM2M APPDATA	19	此LwM2M对象提供LWM2M 服务器相关的应用业务数据，例如水表数据。

说明

OMA组织定义的其它常用对象，详见**http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html**。

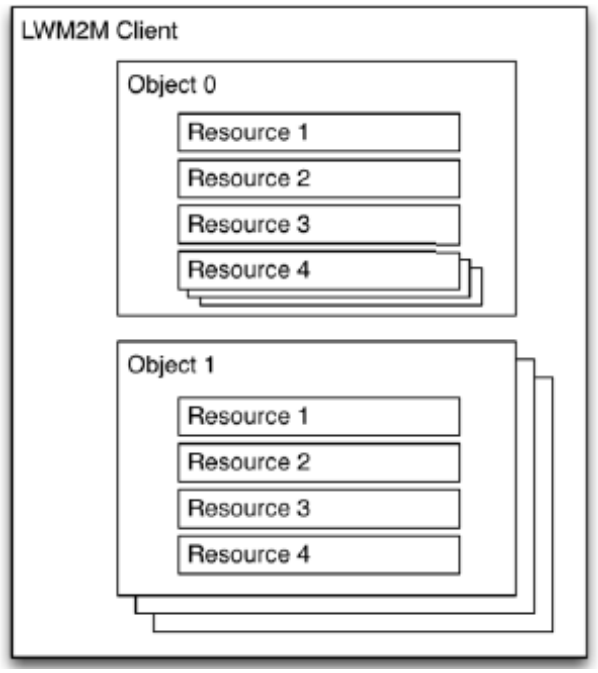
4.1.4.5 LwM2M 资源定义

资源模型

LwM2M定义了一个资源模型，所有信息都可以抽象为资源以提供访问。资源是对象的内在组成，隶属于对象，LwM2M客户端可以拥有任意数量的资源。和对象一样，资源也可以有多个实例。

LwM2M客户端、对象以及资源的关系如图所示：

图 4-8 LwM2M 客户端、对象、资源关系图



资源定义的格式

ID	Name	Operations	Instance	Mandator	Type	Range or Enumeratio	Units	Description
0	Resource Name	R (Read), W (Write), E (Execute)	Multiple/ Single	Mandatory/ Optional	String, Integer, Float, Boolean, Opaque, Time, Objlnk none	If any	If any	Description

4.1.4.6 LwM2M 接口定义

概述

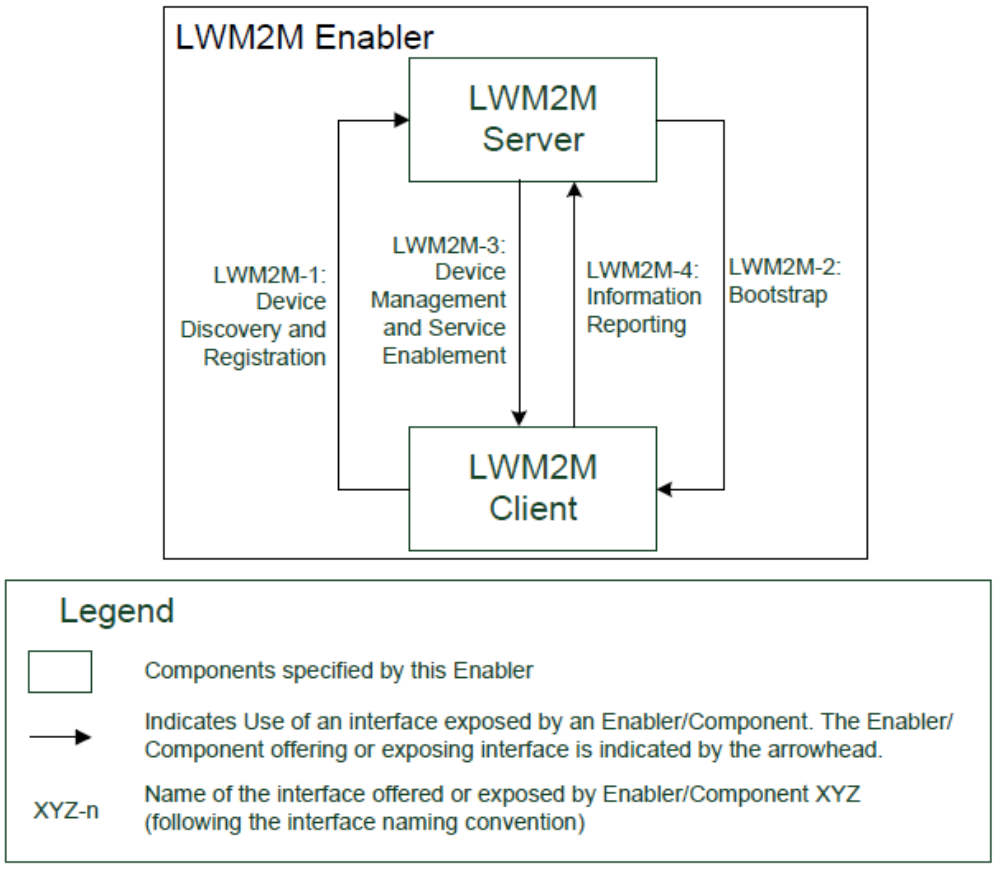
LwM2M引擎主要有两个组件：LwM2M服务器和LwM2M客户端。LwM2M标准为两个组件之间的交互设计了4种主要的接口：

- 设备发现和注册
- 引导程序
- 设备管理和服务实现
- 信息上报

接口模型图

LwM2M接口模型如图所示：

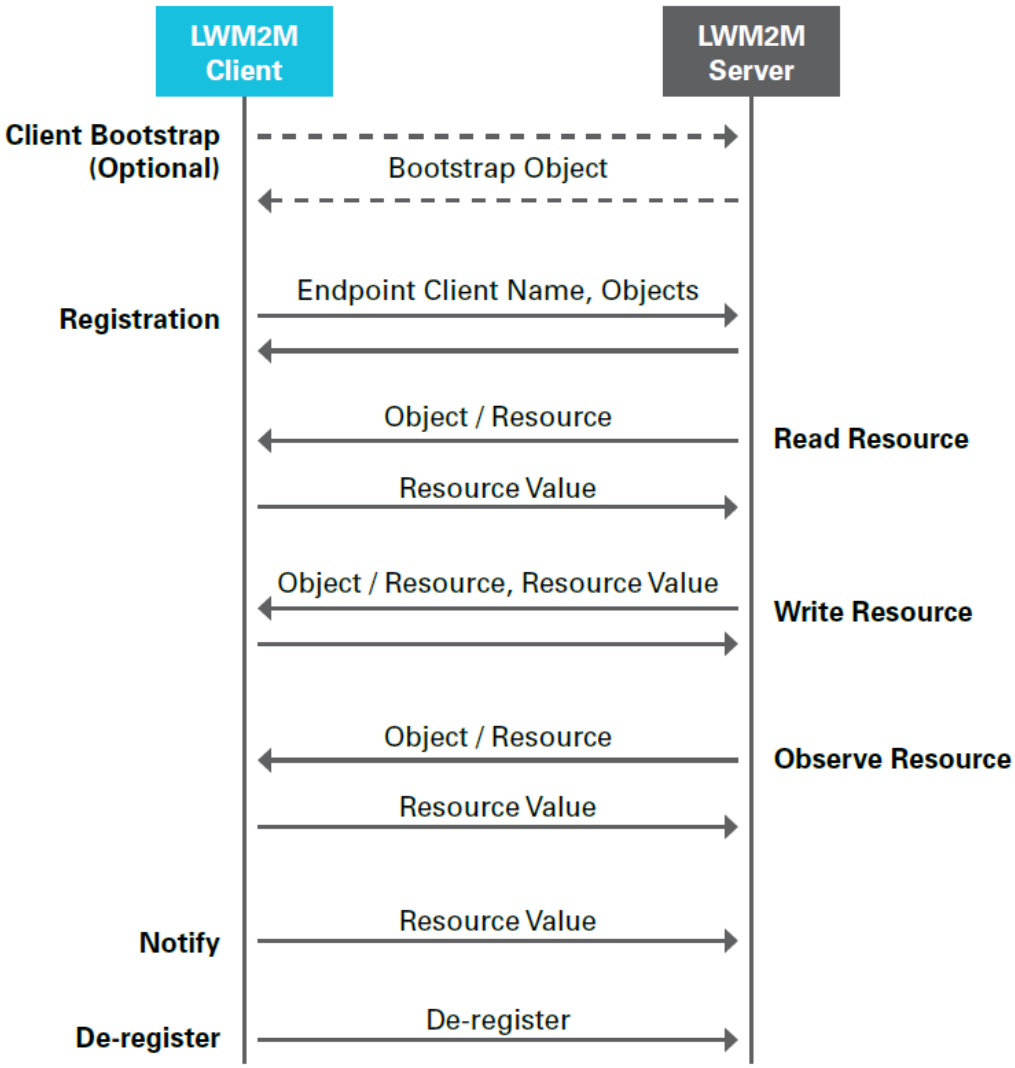
图 4-9 LwM2M 接口模型



消息流程示例

LwM2M的消息交互流程如图所示：

图 4-10 LwM2M 消息流程



设备管理和服务实现接口

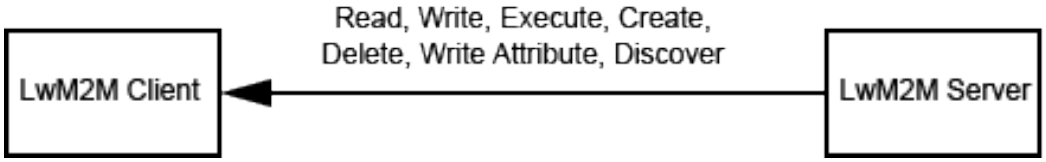
LwM2M的接口表示一类功能，**设备管理和服务实现**接口是LwM2M的四种接口之一。

接口的具体功能是由一系列的操作来实现的，LwM2M的4种接口被分为上行操作和下行操作。

- 上行操作：LwM2M Client -> LwM2M Server
- 下行操作：LwM2M Server -> LwM2M Client

LwM2M Server使用**设备管理和服务实现**接口来访问LwM2M Client的对象实例和资源。该接口包括7种操作：“Create”、“Read”、“Write”、“Delete”、“Execute”、“Write Attributes”和“Discover”。

图 4-11 设备管理和服务实现接口操作



接口	操作	方向
设备管理和服务实现	Create，Read，Write，Delete，Execute，Write Attributes，Discover	下行

设备管理和服务实现接口的交互过程如图所示：

图 4-12 设备管理&服务使能接口示例

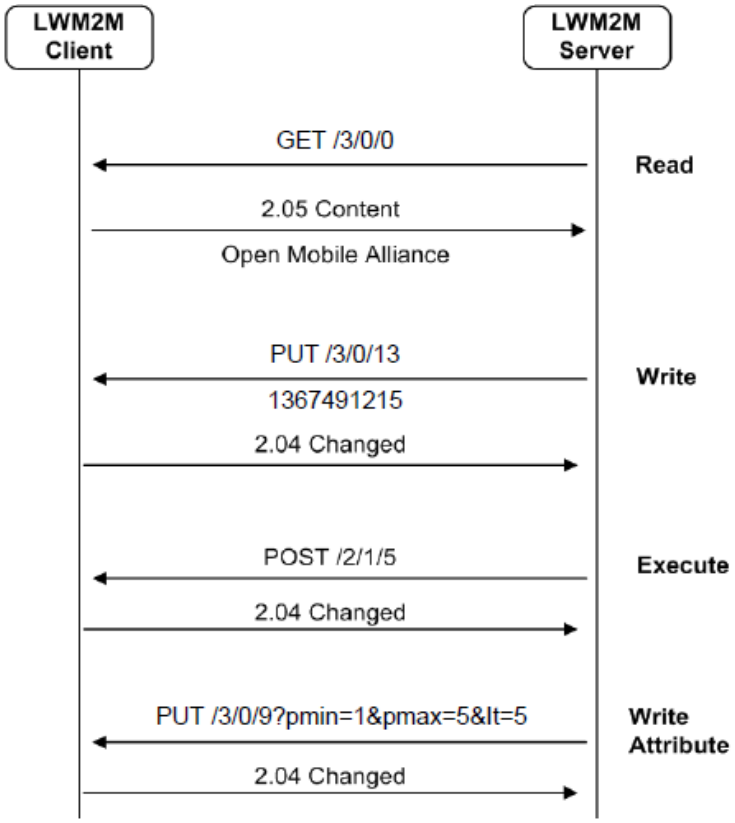
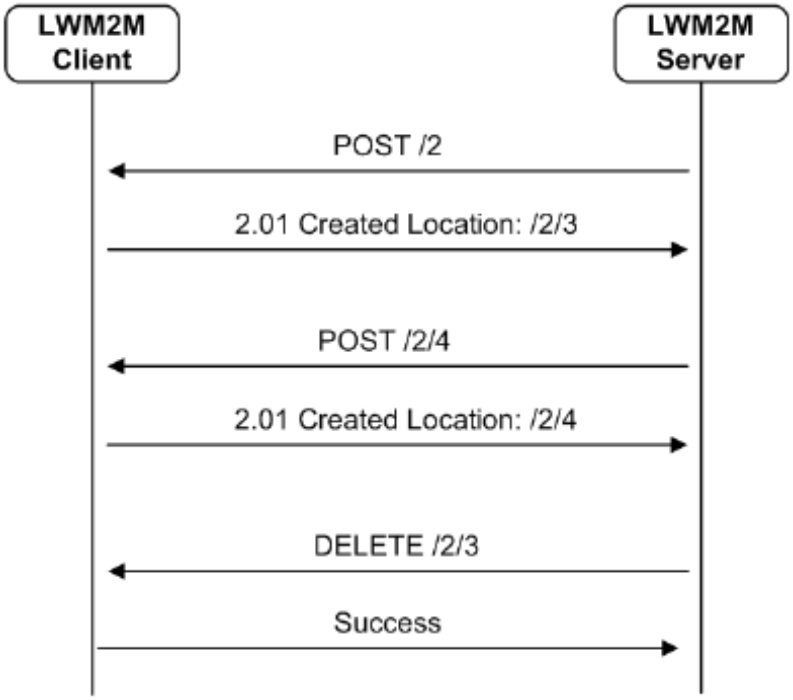


图 4-13 对象创建和删除示例



4.1.4.7 固件升级

LwM2M的固件升级对象使得固件升级的管理成为可能。固件升级对象包括安装固件包、更新固件、以及更新固件之后执行的其它动作。成功进行了固件升级后，device必须要重启，以使新的固件生效。

在设备重启之后，如果“Packet”资源包含了一个合法的但还未被安装的固件包，“State”资源必须为<Downloaded>状态， 否则须为<Idle>状态。

在设备重启之前，标识更新结果的相关数值必须保存。

固件升级对象定义

Name	Object ID	Instances	Mandatory	Object URN
Firmware Update	5	Single	Optional	rn:oma:LwM2M:oma:5

固件升级对象的资源定义

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
0	Package	W	Single	Mandatory	Opaque	-	固件包。

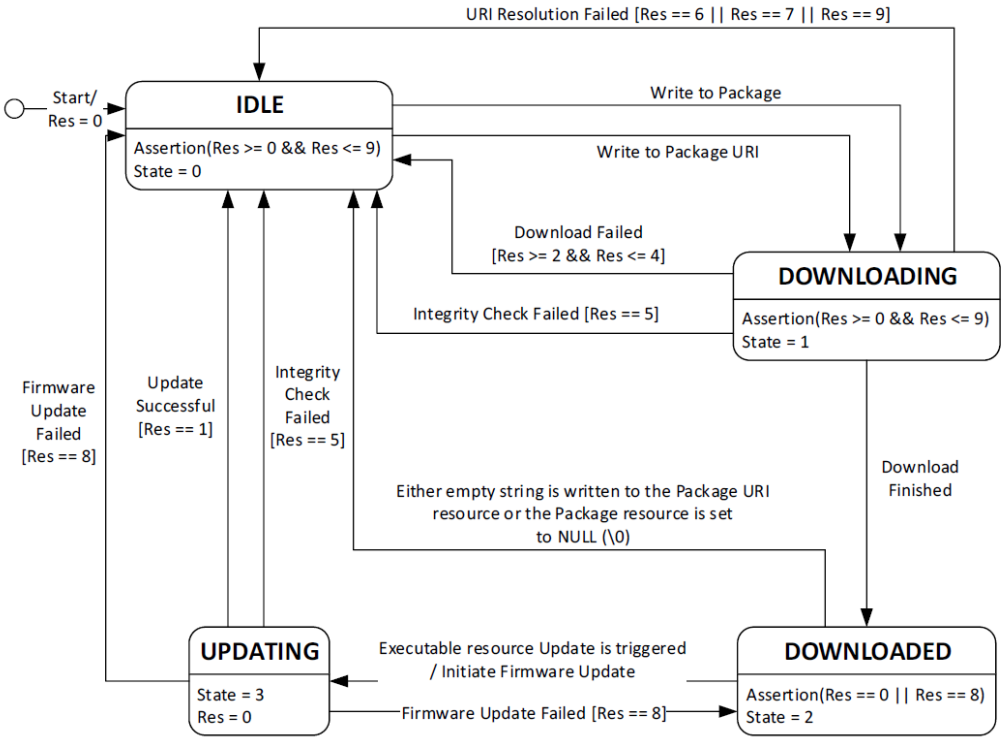
ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
1	Package URI	W	Single	Mandatory	String	0-255 bytes	固件包的下载URI。
2	Update	E	Single	Mandatory	none	no argument	更新固件。 只有当State资源是Downloaded状态时，该资源才是可执行的。
3	State	R	Single	Mandatory	Integer	0-3	固件升级的状态。这个值由客户端设置。0: Idle (下载之前或者成功更新之后) 1: Downloading (下载中) 2: Downloaded (下载已完成) 3: Updating (更新中)在Downloaded状态下，如果执行Resource Update操作，状态则切换为Updating。 如果更新成功，状态切换为Idle；如果更新失败，状态切换回Downloaded。
4	Update Supported Objects	RW	Single	Optional	Boolean	-	默认值是false。 如果该值置为true，则固件成功更新了之后，客户端必须通过更新消息或注册消息，向服务器通知Object的变动。 如果更新失败了，Object参数通过下一阶段的更新消息报告。

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
5	Update Result	R	Single	Mandatory	Integer	0-8	下载/更新固件的结果： 0: 初始值。一旦开始更新流程（下载/更新）开始，该资源的值必须被置为0。 1: 固件升级成功 2: 没有足够空间存储新固件包 3: 下载过程中内存不足 4: 下载过程中连接丢失 5: 新下载的包完整性检查失败 6: 不支持的包类型 7: 不合法的URI 8: 固件升级失败该资源可以通过Observe操作上报。
6	PkgName	R	Single	Optional	String	0-255 bytes	固件包的名字。
7	PkgVersion	R	Single	Optional	String	0-255 bytes	固件包的版本。

固件升级状态机

固件升级状态之间的变换关系如图所示：

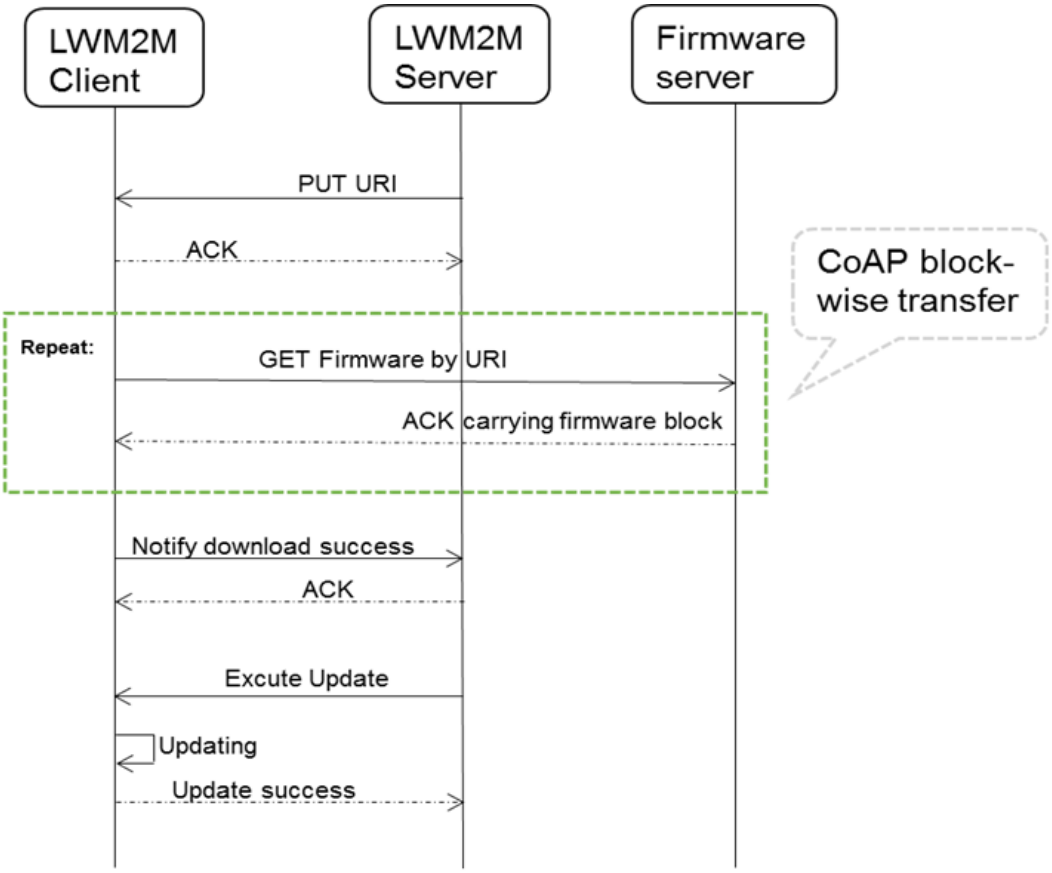
图 4-14 固件升级状态图



固件升级流程

固件升级流程如图所示：

图 4-15 固件升级流程



5 应用侧 SDK 使用指南

JAVA SDK使用指南

5.1 JAVA SDK 使用指南

5.1.1 开发者必读

- 本文以提供的北向Java SDK Demo为例说明如何使用JAVA SDK与IoT平台对接，包括证书配置及回调等。
- Demo以Java工程为例，每个类（除工具类外）都包含了main方法，可单独运行，旨在演示如何调用SDK接口。

5.1.2 开发环境要求

开发环境要求

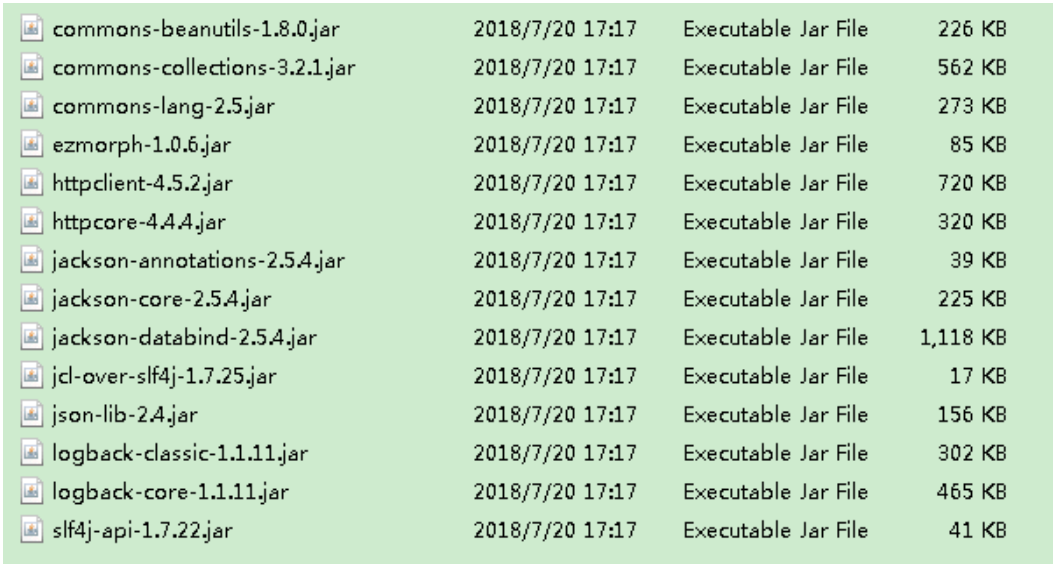
开发平台	开发环境	配套要求	推荐的操作系统
IoT	1) J2EE for Java Developers 2) **Maven**插件：m2e - Maven Integration for **Eclipse** (includes Incubating components)	**JDK 1.8**及以上版本	Windows7

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

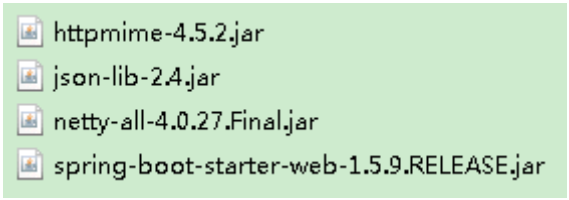
5.1.3 下载相关开发资源

在**资源获取**下载北向JAVA SDK Demo及北向JAVA SDK。

- SDK放在lib目录下，SDK依赖的jar包放在\testSDK\api-client-test_lib下，开发者也可从maven仓库下载。



- Demo工程依赖的jar包放在components文件夹下，开发者也可从maven仓库下载。



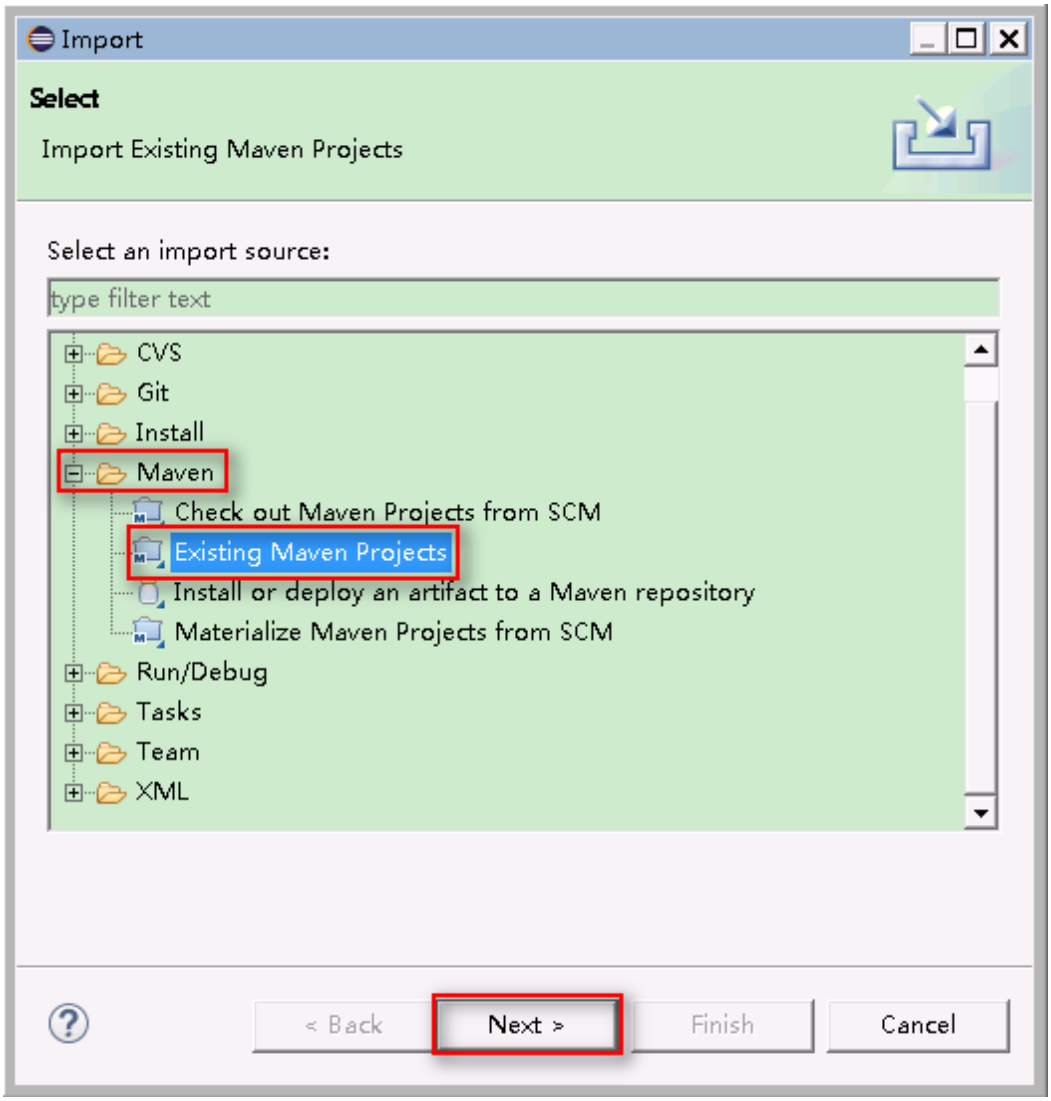
说明

Demo工程中的lib下已包含SDK库。

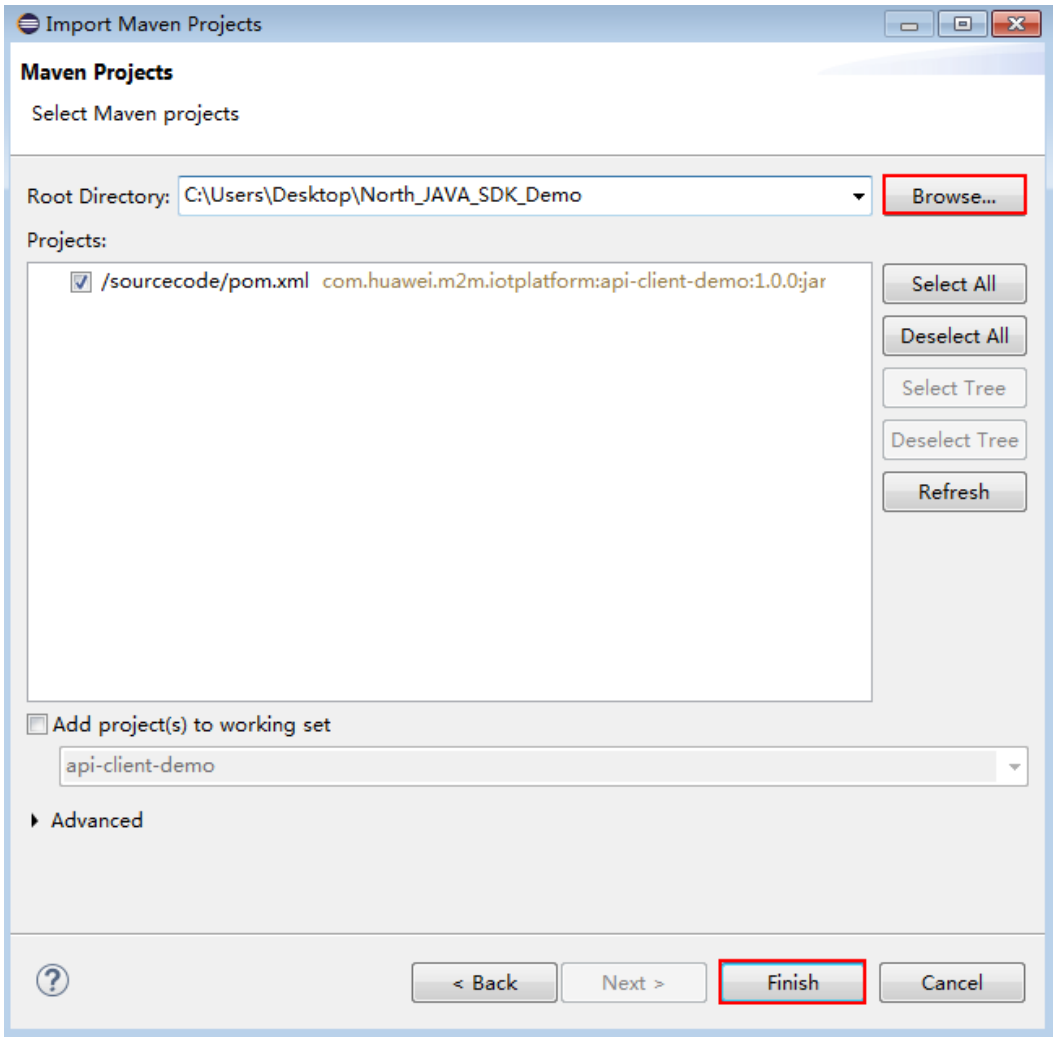
5.1.4 导入 Demo 工程

步骤1 将下载的OceanConJavaDemo.zip解压到本地。

步骤2 打开eclipse，选择菜单“File > Import ”，再选择“Maven > Existing Maven Projects”，单击“Next”。



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



----结束

5.1.5 初始化及证书配置

新建一个**NorthApiClient**实例，设置好**ClientInfo**（包括平台IP、端口、appId和密码），再初始化证书。

注意

- 平台IP、端口、appId和密码都是从配置文件./src/main/resources/application.properties中读取的，因此，当这些信息发生变化时，只要修改配置文件，不用修改应用服务器的代码。
- 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

如果使用测试证书：

```
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
```

使用指定证书

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"));
slConfig.setTrustCAPwd(getAesPropertyValue("newCaPassword"));
slConfig.setSelfCertPath(PropertyUtil.getProperty("newClientCertFile"));
slConfig.setSelfCertPwd(getAesPropertyValue("newClientCertPassword"));

northApiClient.setClientInfo(clientInfo);
northApiClient.initSSLConfig(sslconfig); //使用指定的证书，且默认使用严格主机名校验
```

使用指定证书时，如果不使用严格主机名校验，在调用**northApiClient.initSSLConfig(sslconfig)**之前可以自行设置主机名校验方法：

```
northApiClient.setHostnameVerifier(new HostnameVerifier() {
    public boolean verify(String arg0, SSLSession arg1) {
        // 自定义主机名校验
        ……
        return true;
    }
});
```

注意

主机名校验方法应以安全为原则，不应该直接返回true。

5.1.6 业务接口调用方法

在初始化及证书配置中设置好**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。

鉴权

```
//得到NorthApiClient实例后，再使用northApiClient得到鉴权类实例
Authentication authentication = new Authentication(northApiClient);

//调用鉴权类实例authentication提供的业务接口，如getAuthToken
AuthOutDTO authOutDTO = authentication.getAuthToken();

//从返回的结构体authOutDTO中获取需要的参数，如accessToken
String accessToken = authOutDTO.getAccessToken();
```

订阅

```
//得到NorthApiClient实例后，再使用northApiClient得到订阅类实例
SubscriptionManagement subscriptionManagement = new SubscriptionManagement(northApiClient);

//先设置好subDeviceData的第一个入参SubDeviceDataInDTO结构体
SubDeviceDataInDTO sddInDTO = new SubDeviceDataInDTO();
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();
```

说明

关于哪些参数需要设置，请查看《北向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。

5.1.7 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
}
```

说明

- 接收到平台推送的消息后，开发者需要根据业务进行处理，但不建议进行复杂计算、I/O操作或者可能长时间等待的动作，可以先写数据库，应用进入相应界面或者刷新界面再从数据库取数据并进行数据处理。
- 回调路径已在SDK中设置好了，所以在订阅时要注意订阅对应的回调地址，具体可参考《北向JAVA SDK API参考》文档中消息推送章节的接口。
- 回调的IP地址则是服务器的地址，需要是公网地址。
- Demo工程的回调端口配置在src\main\resource\application.properties中：

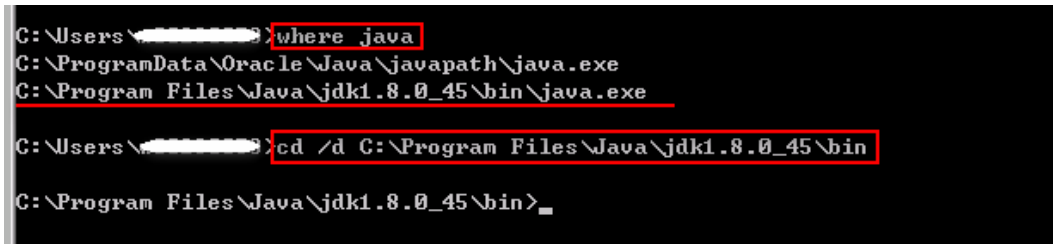
  ```
  #specify the port of the web application
  server.port=8099
  ```

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}
```



步骤2 使用如下命令生成tomcat.keystore文件。

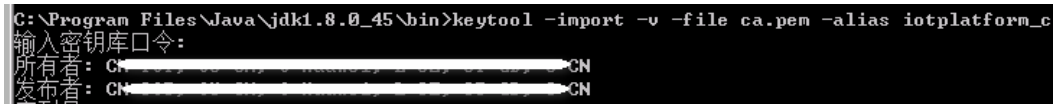
```
keytool -genkey -v -alias tomcat -keyalg RSA -keystore tomcat.keystore -validity 36500
```

注意

- 如果jdk的bin目录下已有tomcat.keystore，建议先将已有的tomcat.keystore移到别的路径下。
- “您的名字与姓氏是什么”要输入应用服务器的IP或域名。
- <tomcat>的密钥口令要与密钥库口令设置一致（最后一步按回车即可），**输入的密钥库口令要记住，后续配置会使用到。**

步骤3 将IoT平台提供的根证书ca.pem放到jdk的bin目录下，并使用如下命令将其加到tomcat.keystore的信任证书链中。

```
keytool -import -v -file ca.pem -alias iotplatform_ca -keystore tomcat.keystore
```



输入密钥库口令，查看导入的证书内容，确认无误后，输入y即可。

说明

- 平台的测试根证书ca.pem可以在JAVA SDK包的cert目录下找到。
- 将IoT平台提供的根证书ca.pem加到tomcat.keystore的信任证书链后，由ca.pem签发的子证书就能得到应用服务器的信任。

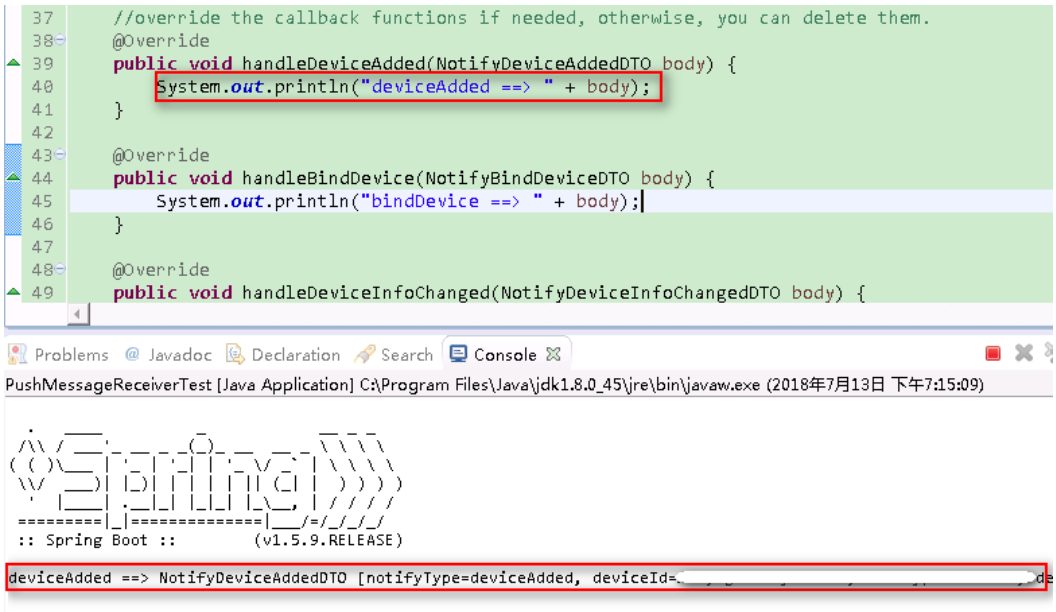
步骤4 将tomcat.keystore放到Demo工程目录下，例如：src\main\resources，打开src\main\resource\application.properties，添加如下配置。其中，server.ssl.key-store 为tomcat.keystore 所在路径，server.ssl.key-store-password为密钥库口令。

```
#one-way authentication (server-auth)
server.ssl.key-store=./src/main/resources/tomcat.keystore
server.ssl.key-store-password=741852963.
```

步骤5 右键单击**PushMessageReceiverTest**，选择“Run As > Java Application ”，运行Demo中的**PushMessageReceiverTest**类。运行结果如下：

说明

当有数据推送到应用服务器时，就会进入相应的回调函数中。



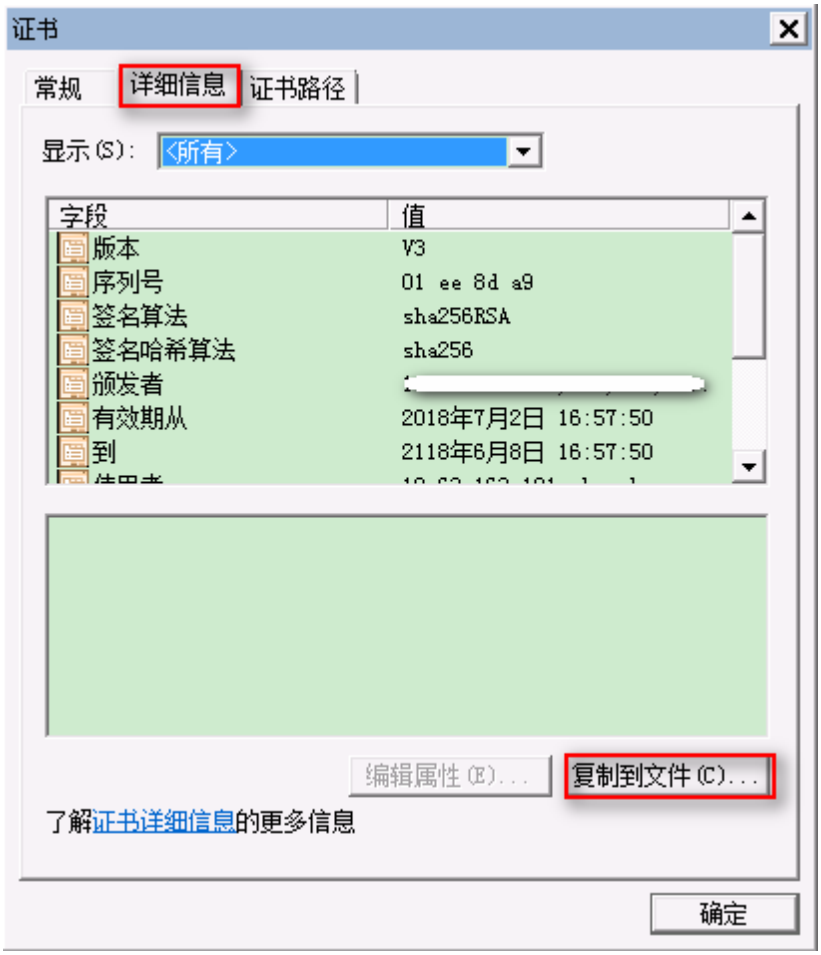
----结束

回调证书导出

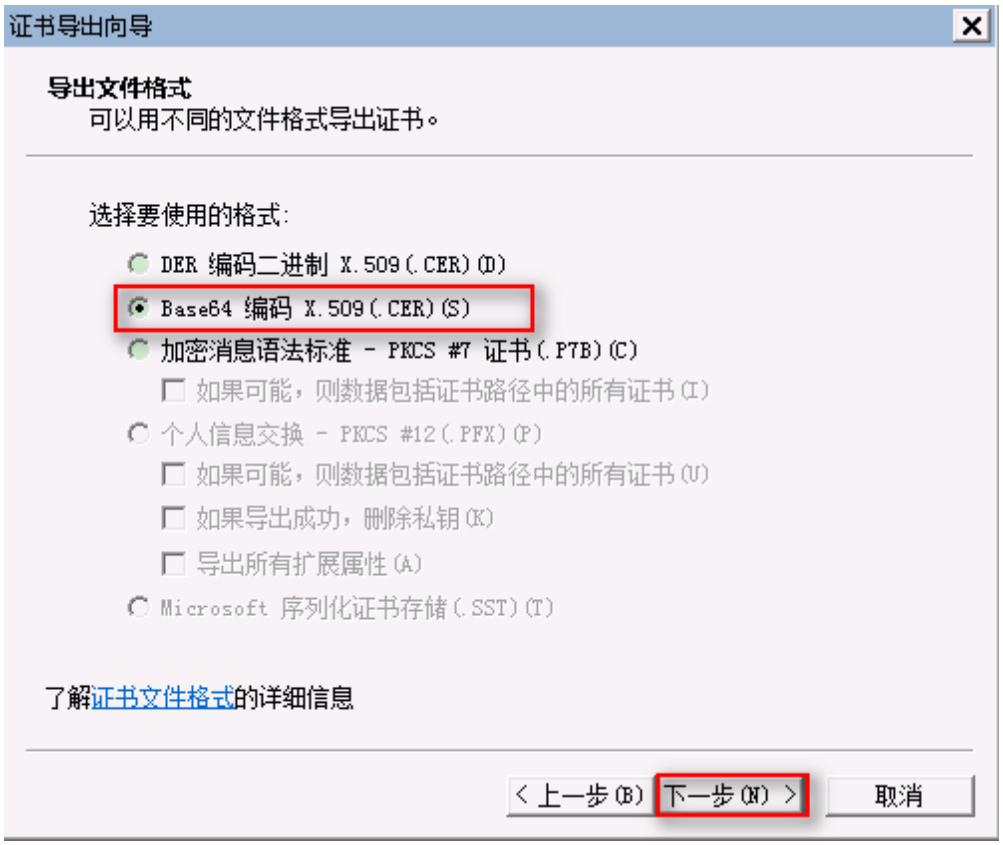
步骤1 使用浏览器打开回调地址https://server:8099/v1.0.0/messageReceiver，以Google为例，并查看证书。

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”。

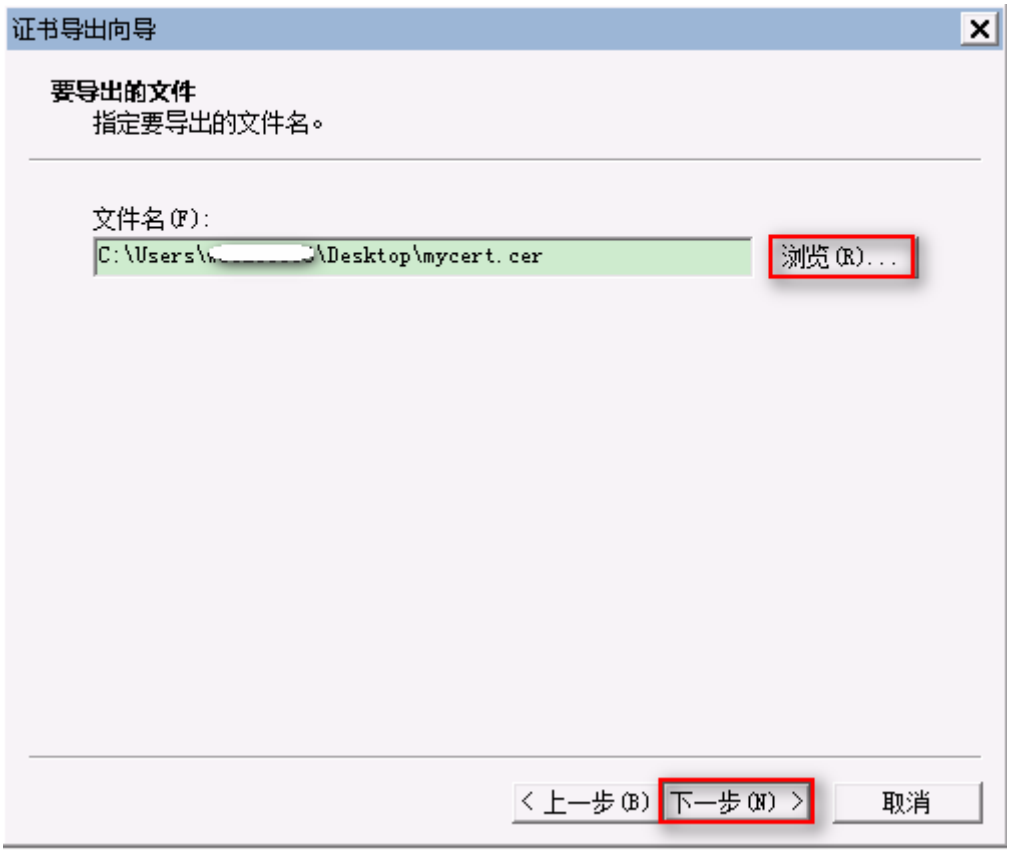


步骤3 单击“下一步”，进入“导出文件格式”界面，选择“Base64编码”，然后单击“下一步”。

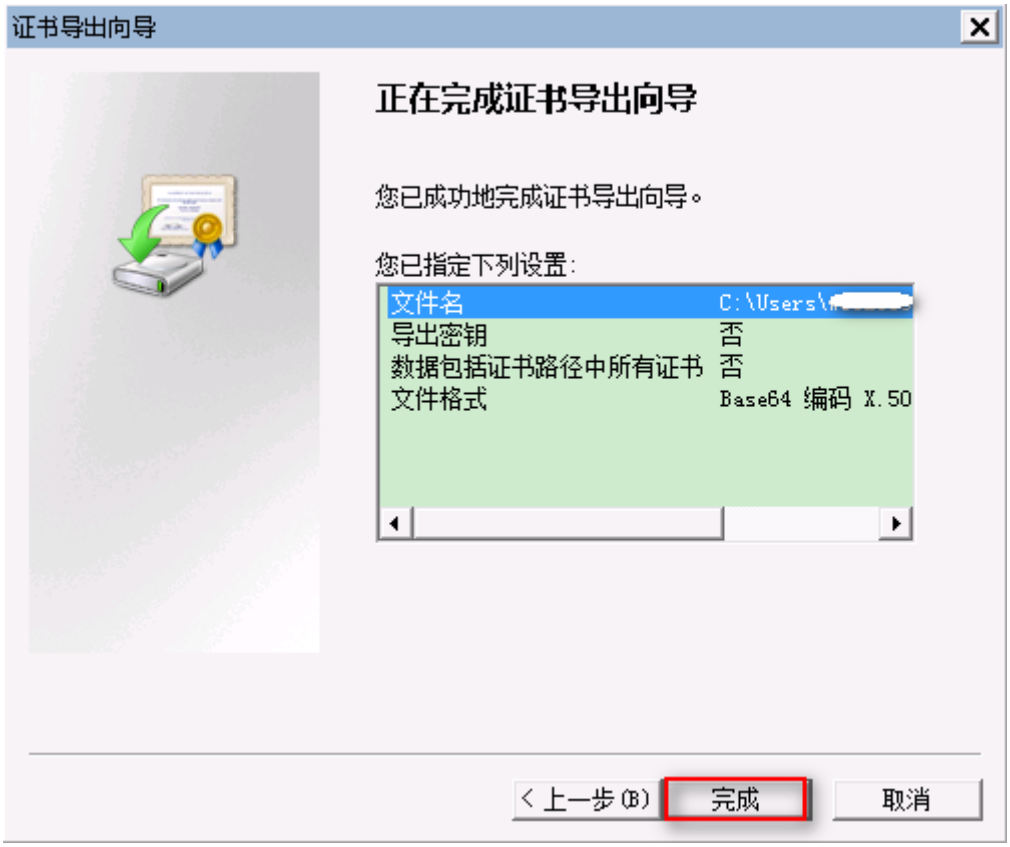


步骤4 指定证书的保存路径，完成证书导出。

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。

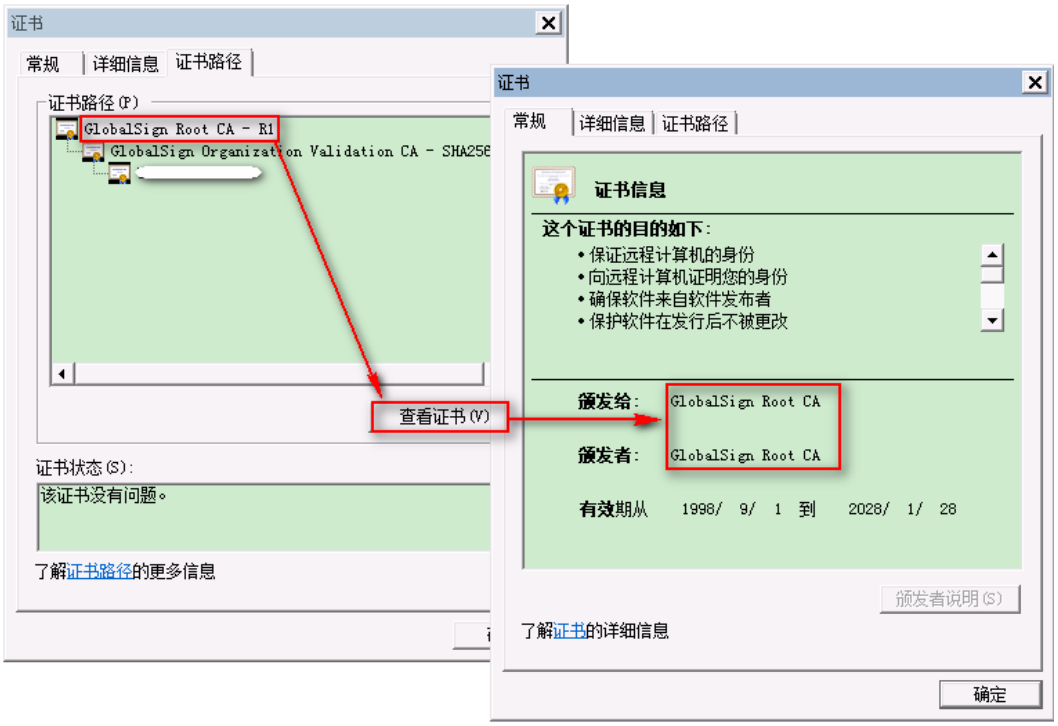


注意

如果应用服务器最后部署到云上，可能会有多级证书，**建议在部署完成后再导出证书**。此时需要将证书链**上面几级的证书一一导出**。

步骤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IoT平台相应的应用下。

```
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
```

----结束

注意

- Demo中的配置为单向认证，在**导出证书后**需要修改为双向认证。将以下配置打开（去掉注解，修改tomcat.keystore目录与密码），由于已将平台的根证书加入tomcat.keystore的信任证书链中，所以不需要再做其他修改，重启一下服务器即可。

```
#two-way authentication (add client-auth)
server.ssl.trust-store=./src/main/resources/tomcat.keystore
server.ssl.trust-store-password=741852963.
server.ssl.client-auth=need
```

- 单向认证较之双向认证安全度低，请使用双向认证。

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”。

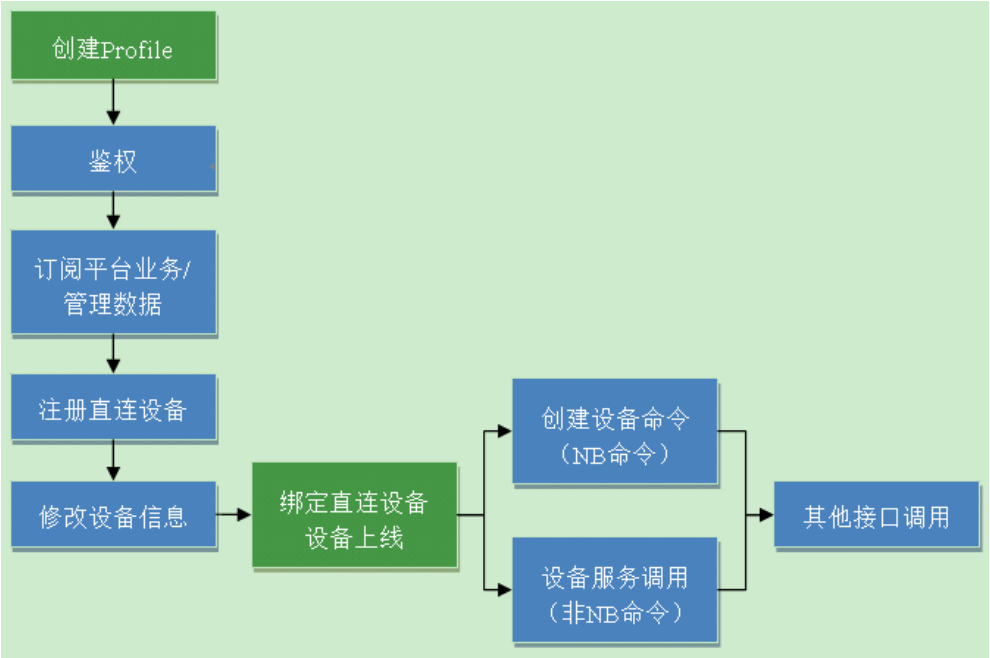
步骤3 单击“添加”，上传证书。

----结束

5.1.8 业务接口调用流程及注意事项

业务接口调用方法一节已说明如何调用接口，其他的接口调用方法与之类似，请直接参考Demo代码，不再逐一说明。

- 请按如下流程调用业务接口。



- 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”。

  说明

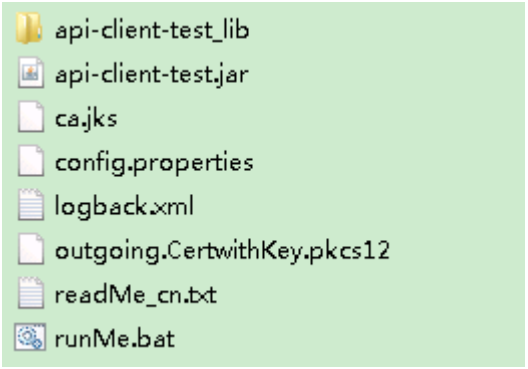
  建议在定义好Profile后再调用接口注册设备。

- 修改设备信息接口使用到的字段值如“设备类型”、“厂商名”、“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《JAVA SDK API参考文档》中“应用安全接入 > 定时刷新token ”章节的说明。

5.1.9 SDK 独立运行测试

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

图 5-1 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 5-2 修改 config.properties

```
#please modify the value of platformIp/platformPort/
platformIp=1[redacted]
platformPort=8743
appId=xo1D12[redacted]MWE5a8DIa
secret=gPnTW[redacted]kkf12P8f4a
#the value of newCaFile and newClientCertFile should
newCaFile=
newCaPassword=
newClientCertFile=
newClientCertPassword=
#hostNameVerify default value is true, true means se
hostNameVerify=
```

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[redacted]DK
[Y] getAuthToken() 1, get accesstoken successfully with inner certificates
AuthOutDTO [accessToken=57a37baa92cbbfca3d763ffe8e9e9d73, tokenType=bearer, refreshToken=4ba44a
[Y] refreshAuthToken() succeeded, AuthRefreshOutDTO [accessToken=309c198bc768ca31eb83996936fba2
3762bcf2]
[Y] regDirectDevice() succeeded, DirectDeviceRegOutDTO [deviceId=ef72daa2-772d-4a9b-a2bc-71a48a

[Y] modifyDeviceInfo() succeeded
[Y] deleteDevice() succeeded
```

----结束