ALIBABA CLOUD

阿里云

阿里云最佳实践

物联网设备上云

文档版本：20210527

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.无操作系统设备通过TCP模组上云

无操作系统的MCU设备通过TCP模组接入物联网平台，实现数据上报。

前提条件

在进行本实践前，您需要完成以下准备工作：

- 注册阿里云账号，并完成实名认证。
- 开通物联网平台，点击查看物联网平台详情页。
- 准备MCU开发板及软件开发环境。

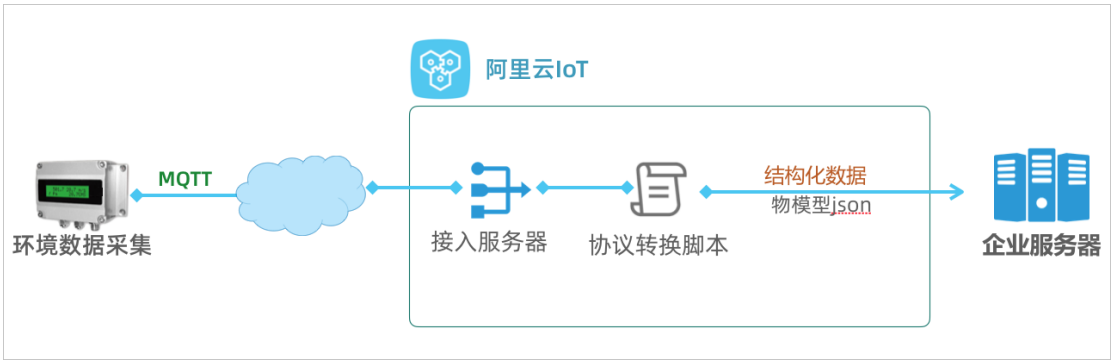
> 说明　本文中使用以下设备进行开发实践。
>
> - 开发环境为IAR Embedded Workbench for ARM，点击查看详情。
> - MCU为ST公司（意法半导体公司）生产的STM32F103，点击查看详情。
> - 开发板NUCLEO-F103RB，点击查看详情。
> - 通信模组为SIMCom公司（芯讯通无线科技有限公司）生产的SIM800C，点击查看详情。
> - 开发板SIM800C mini V2，您可以点击登录淘宝购买。

背景信息

阿里云物联网平台官方发布的C语言版本的设备端SDK 可以接入到无操作系统的MCU（Micro controller）上，使用MQTT协议完成设备上云。点击查看设备端SDK详情。

无操作系统设备通过TCP模组上云的流程图如下图所示。

上云流程图



创建产品和设备

在物联网平台创建产品和设备，获取设备证书信息。在物联网平台注册产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需配置到设备端SDK中。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

1. 点击登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

iii. 填入产品信息，单击**确定**。完成产品创建。

3. 创建设备。

i. 在左侧导航栏，选择**设备**。

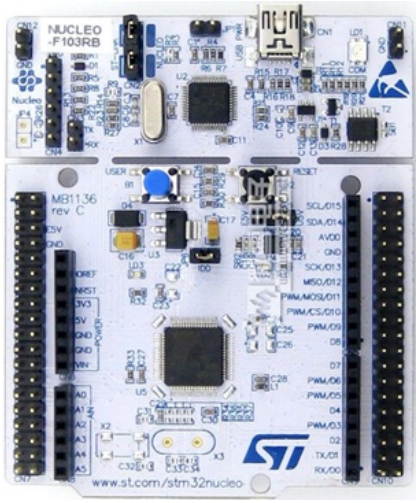
ii. 在**设备管理**页，单击**添加设备**。

iii. 选择刚创建的产品，输入**设备名称**和**备注名称**，单击**确定**。完成设备创建。设备创建成功后，会弹出设备证书信息。您也可以在设备管理页，单击设备对应的查看按钮，进入设备详情页查看设备证书信息。

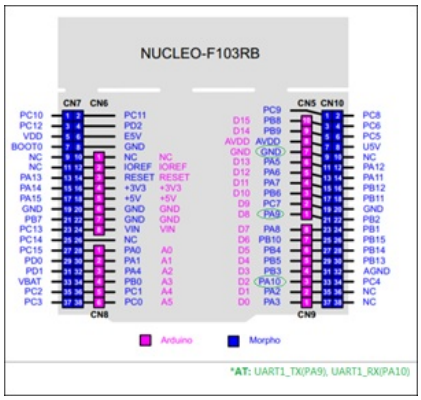
开发设备端

本示例中使用了以下两个开发板：

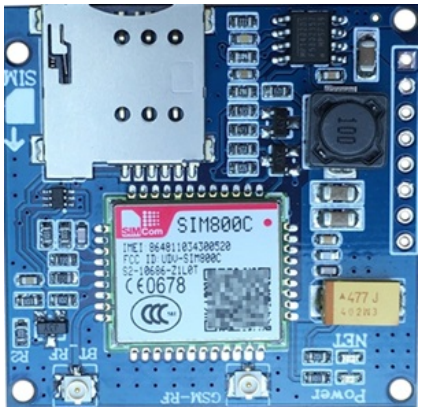
- 开发板NUCLEO-F103RB



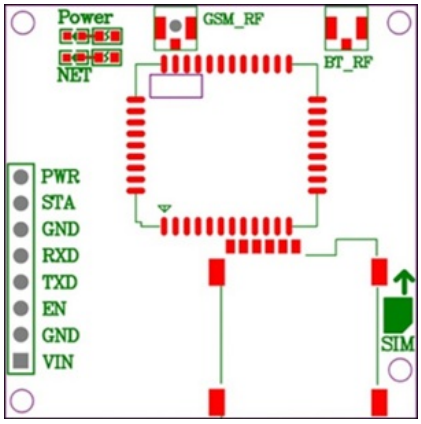
引脚示意图如下。



- MCU是SIM800C mini v2.0

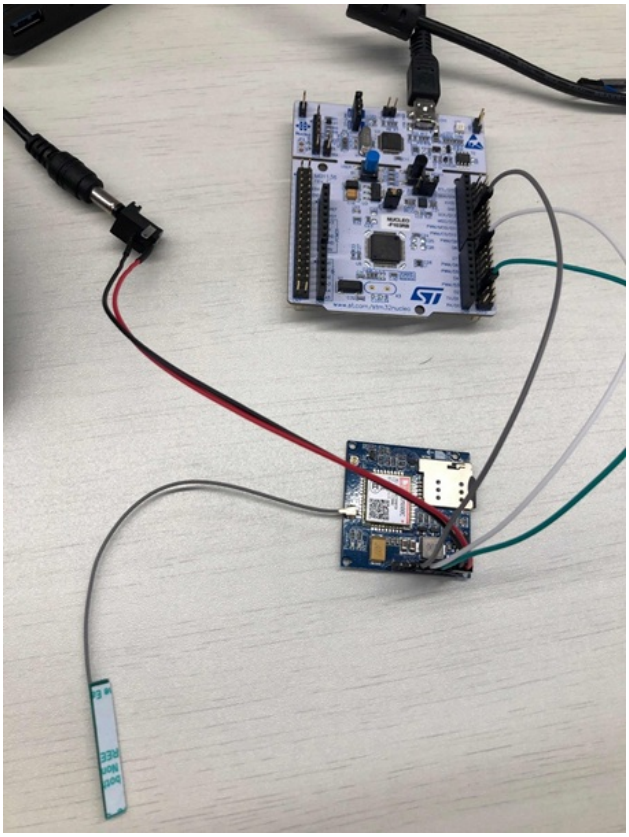


引脚示意图和说明如下。

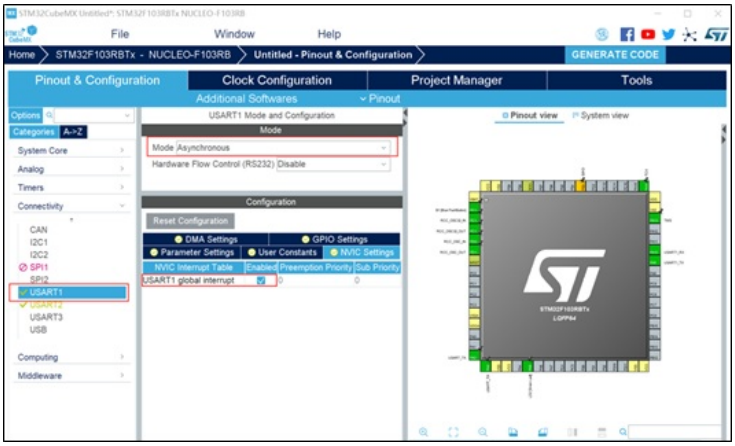


引脚	说明
PWR	开关机引脚。默认为自动开机。
STA	状态监测引脚。
GND	电源接地引脚。
RXD	接收串口引脚。
TXD	发送串口引脚。
EN	电源使能引脚。
VIN	5~18V电源输入。

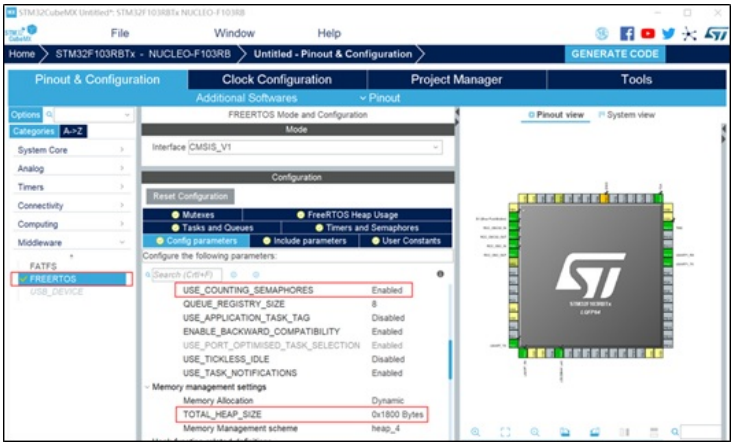
1. 连接硬件。将两个开发板的接收和发送串口连接，作为AT指令通道，如下图所示。



2. 搭建软件环境。
   i. 打开ST-cubemx，新建Project。点击查看ST-cubemx使用详情。
   ii. 在Board Seletor中，选择NUCLEO-F103RB开发板。
   iii. 在Connectivity菜单中，添加串口USART1作为MCU与模组通信的端口，并进行以下配置。
   - 在Configuration中选择USARTx。
   - 将USART1(AT端口)的interrupt设置为enable。



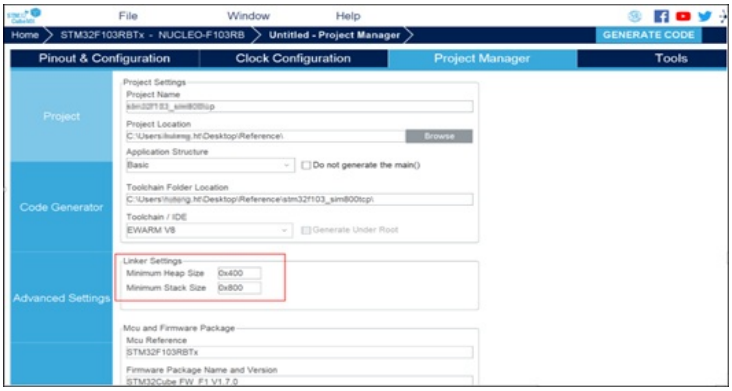
iv. 在Middleware菜单中，选择FREERTOS，并配置为使用计数信号量和堆大小（用于给每个线程分配栈）。



v. 单击Apply>OK。

vi. 单击Cubemx的Project选项中的Generate Code，并进行设置。

- Toolchain/IDE选择为IAR™。
- Heap/Satck size按需进行配置。



vii. 单击OK生成代码工程。

3. 配置设备端SDK。本示例使用的设备端C语言SDK版本为3.0.1。

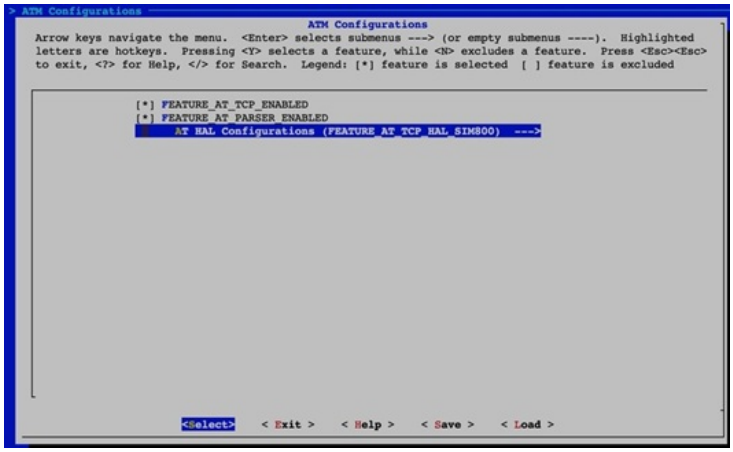
i. 点击下载SDK。

ii. 从下载包中提取SDK代码。本文以Linux系统操作为例。

a. 运行make menuconfig。

b. 选中ATM Configurations，单击Select。

c. 选中AT HAL Configurations，单击Select。



d. 配置如下项目。

```
FEATURE_PLATFORM_HAS_STDINT=y
FEATURE_INFRA_STRING=y
FEATURE_INFRA_NET=y
FEATURE_INFRA_LIST=y
FEATURE_INFRA_LOG=y
FEATURE_INFRA_LOG_ALL_MUTED=y
FEATURE_INFRA_LOG_MUTE_FLW=y
FEATURE_INFRA_LOG_MUTE_DBG=y
FEATURE_INFRA_LOG_MUTE_INF=y
FEATURE_INFRA_LOG_MUTE_WRN=y
FEATURE_INFRA_LOG_MUTE_ERR=y
FEATURE_INFRA_LOG_MUTE_CRT=y
FEATURE_INFRA_TIMER=y
FEATURE_INFRA_SHA256=y
FEATURE_INFRA_REPORT=y
FEATURE_INFRA_COMPAT=y
FEATURE_DEV_SIGN=y
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_MQTT_DEFAULT_IMPL=y
FEATURE_MQTT_DIRECT=y
FEATURE_DEVICE_MODEL_CLASSIC=y
FEATURE_ATM_ENABLED=y
FEATURE_AT_TCP_ENABLED=y
FEATURE_AT_PARSER_ENABLED=y
FEATURE_AT_TCP_HAL_SIM800=y
```

e. 配置完成后，在Linux中运行 `./extract.sh` 提取代码。

```
huteng@r10c05067:~/c-sdk$ ./extract.sh
. Download request sent, waiting respond ...
. Respond generating, wait longer
. Retried 1/20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  107k  100  107k    0     0   408k      0 --:--:-- --:--:-- --:--:--  408k

Please pick up extracted source files in [/disk1/huteng/c-sdk/output]
```

提取的代码位于*output/eng*目录。

```
huteng@r10c05067:~/c-sdk/output/eng$ ls
atm  dev_sign  infra  mqtt  sdk_include.h  wrappers
```

其中，各子目录分别包含的代码如下表。

目录	代码内容
atm	AT指令收发模块
dev_sign	设备身份认证模块
infra	内部实现模块
mqtt	MQTT协议模块
wrappers	HAL对接模块

iii. 在*wrappers*目录下，新建文件*wrappers.c*，该文件中的代码需实现以下HAL函数。

```
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size, uint32_t
*recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size, uint32_t
timeout)
int HAL_GetFirmwareVersion(char *version)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void*HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void*HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void HAL_Printf(const char *fmt, ...)
void HAL_SleepMs(uint32_t ms)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

iv. 在代码Demo中，替换设备证书信息为您的设备证书信息。

```
 * NOTE:
 *
 * HAL_TCP_xxx API reference implementation: wrappers/os/ubuntu/HAL_TCP_linux.c
 *
 */
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "stm32f1xx_hal.h"
#include "infra_types.h"
#include "infra_defs.h"
#include "wrappers_defs.h"
#include "at_wrapper.h"

#define EXAMPLE_PRODUCT_KEY         "a1[redacted]"
#define EXAMPLE_PRODUCT_SECRET      "4R[redacted]Wog2E"
#define EXAMPLE_DEVICE_NAME         "example"
#define EXAMPLE_DEVICE_SECRET       "NR[redacted]j"

#define EXAMPLE_FIRMWARE_VERSION    "app-1.0.0-20190118.1000"
#define RING_BUFFER_SIZE            (128)
typedef struct
{
  uint8_t  data[RING_BUFFER_SIZE];
  uint16_t tail;
  uint16_t head;
}uart_ring_buffer_t;
```

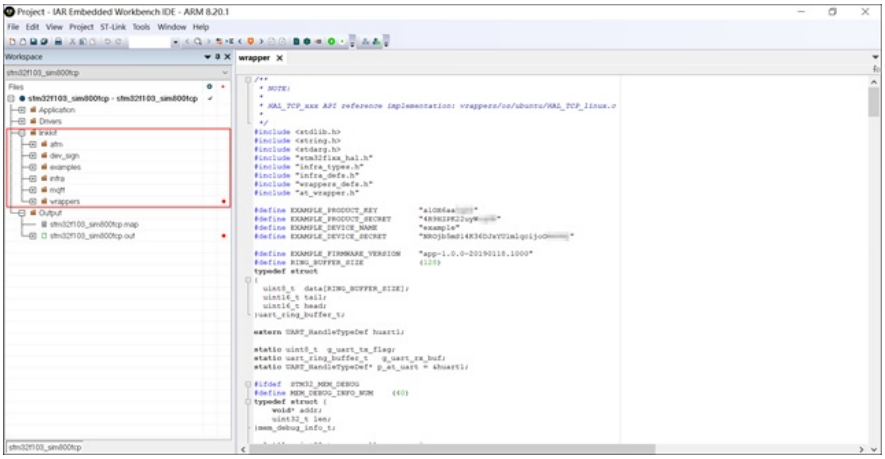
点击下载wrappers.c文件的代码Demo。

> **说明** 如果通信模组为其他模组，则在配置 `FEATURE_AT_TCP_HAL_SIM800=n` 。需实现的HAL函数列表如下所示。

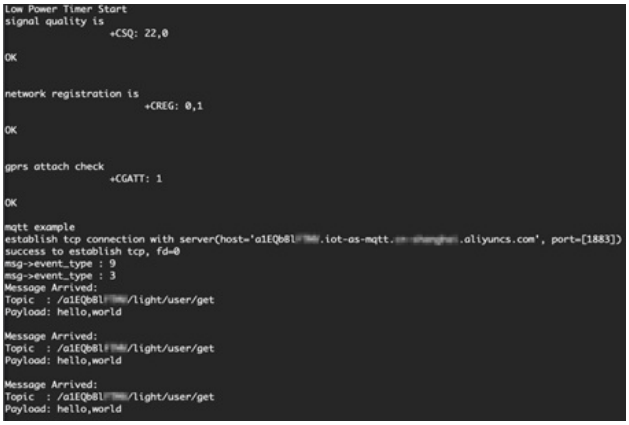
```
int HAL_AT_CONN_Close(int fd, int32_t remote_port)
int HAL_AT_CONN_Deinit(void)
int HAL_AT_CONN_DomainToIp(char *domain, char ip[16])
int HAL_AT_CONN_Init(void)
int HAL_AT_CONN_Send(int fd, uint8_t *data, uint32_t len, char remote_ip[16],int32_t remote_port, int32_t time
out)
int HAL_AT_CONN_Start(at_conn_t *conn)
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size, uint32_t
*recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size, uint32_t
timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN + 1])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN + 1])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN + 1])
void*HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void*HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void HAL_Printf(const char *fmt, ...)
void HAL_SleepMs(uint32_t ms)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

整合SDK

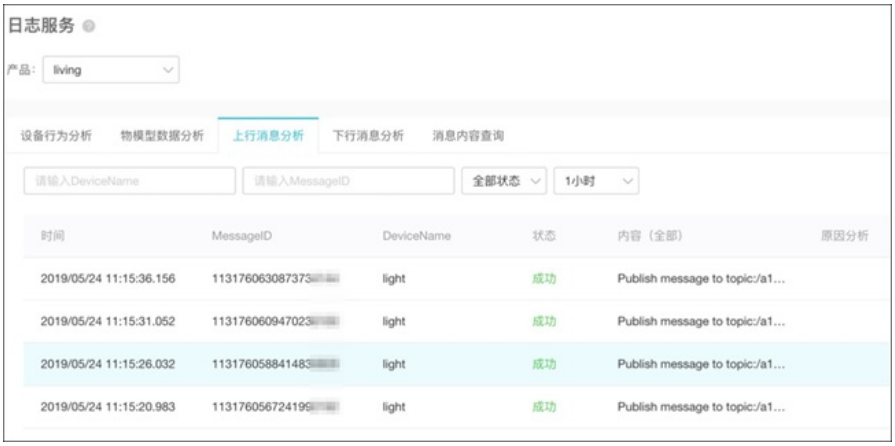
1. 将SDK整合到IAR工程中。如下图所示。



2. 运行SDK。运行SDK，进行测试。运行成功后，设备端日志如下。



在物联网平台控制台，**监控运维>日志服务**中，也可查看设备上报数据到云端的日志。



名词解释

- **MCU**

  微控制单元(Microcontroller Unit)，又称单片机。

- **设备端SDK**

  阿里云物联网平台提供的Link Kit SDK即设备端SDK，用于设备端开发。设备需要支持TCP/IP协议栈才能集成Link Kit SDK。设备厂商将设备端SDK集成到设备上，设备便可通过该SDK安全地接入到阿里云物联网平台。目前，物联网平台提供六种设备端SDK：C SDK、Java SDK、Python SDK、Node.js SDK、Android SDK和iOS SDK。本示例使用的是C SDK。

  点击下载SDK并查看使用说明。

- **设备证书**

设备证书指ProductKey、DeviceName和DeviceSecret，是阿里云物联网平台认证设备的标识。设备证书信息不可泄露。

- **ProductKey**

  物联网平台为产品颁发的全局唯一标识。

- **DeviceName**

  在注册设备时，自定义的或系统自动生成的设备名称，具备产品维度内的唯一性。

- **DeviceSecret**

  物联网平台为设备颁发的设备密钥。

更多最佳实践

点击查看更多阿里云最佳实践。

2.温湿度采集设备以HTTPS上云

本文以温湿度采集器为例，介绍设备通过HTTPS协议连接物联网平台并上报数据的配置和开发方法。

前提条件

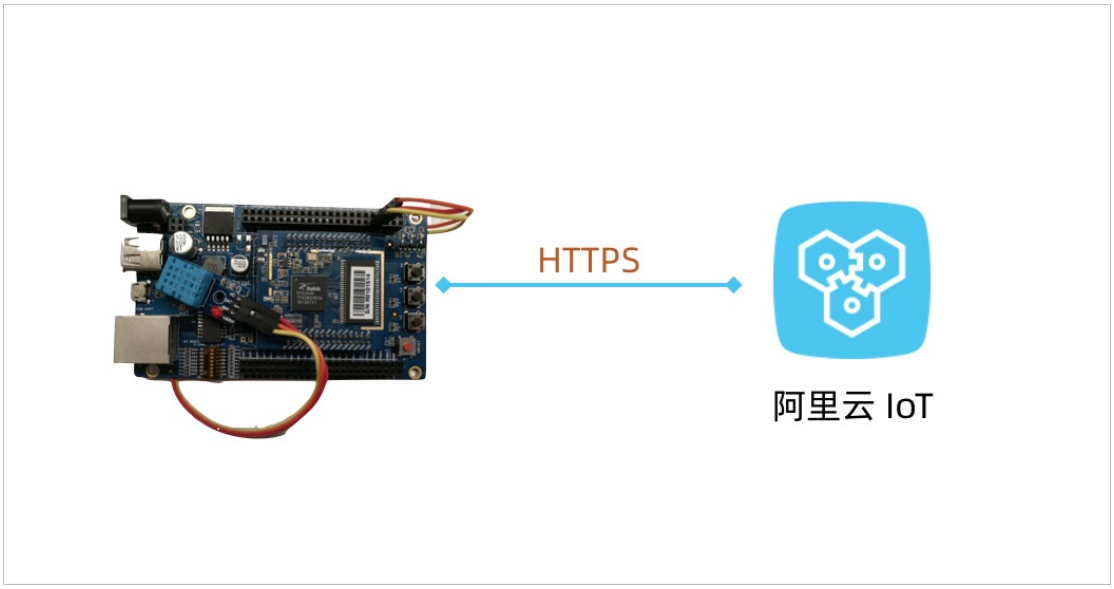
在进行本示例配置前，您需要完成以下准备工作：

- 注册阿里云账号，并完成实名认证。
- 开通物联网平台。关于物联网平台的介绍，参见物联网平台产品详情页。
- 本文示例需准备Node.js 6及以上开发环境。

背景信息

物联网平台华东2（上海） 地域支持设备使用HTTPS协议接入。设备与物联网平台通过HTTPS协议进行连接通信仅适用于单纯的设备上报数据场景。请求方式仅支持POST，且设备上报的数据不超过128 KB。

温湿度采集设备上云架构图



创建产品和设备

在物联网平台创建产品和设备后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需烧录到设备固件上。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

1. 点击登录物联网平台控制台。
2. 选择地域为**华东2（上海）**。

3. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

iii. 填入产品信息，单击**确定**。完成产品创建。

4. 创建设备。
   i. 在左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。
   iii. 选择刚创建的产品，输入设备名称和备注名称，单击**确定**。完成设备创建。设备创建成功后，会弹出设备证书信息。您也可以在**设备管理**页，单击设备对应的**查看**按钮，进入**设备详情**页查看设备证书信息。

定义物模型

物模型指将物理空间中的实体进行数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能（包括属性、事件、服务）。完成功能定义后，系统将自动生成该产品的物模型。本示例中，温湿度采集器会上报温度和湿度，因此需为该产品定义对应的两个属性。

1. 在物联网平台控制台的左侧导航栏，选择**设备管理>产品**。
2. 在**产品管理**页，找到之前创建的产品，单击对应的**查看**按钮。
3. 在**产品详情**页，选择**功能定义**，再单击自定义功能对应的**添加功能**。
4. 根据下表逐个添加温度和湿度属性。

功能类型	功能名称	标识符	数据类型	取值范围	步长	读写类型
属性	温度	temperature	int32	-10~50	1	只读
属性	湿度	humidity	int32	1~100	1	只读

设备端开发

设备端开发需实现设备通过HTTPS协议连接物联网平台，并上报温湿度属性数据。点击查看HTTPS连接通信文档。

1. 认证设备身份。设备请求物联网平台建立连接时，物联网平台进行设备身份认证。认证通过后，下发设备token。设备token将在设备上报数据时使用。设备身份认证请求参数如下表。

参数	说明
method	请求方法。必须指定为POST。
uri	指定为：iot-as-http.cn-shanghai.aliyuncs.com/auth。
productKey	设备所属产品的Key。可从物联网平台的控制台设备详情页获取。

参数	说明
deviceName	设备名称。从物联网平台的控制台设备详情页获取。
clientId	客户端ID。长度为64字符内，可使用设备的MAC地址或SN码。本示例中，使用函数random()生成随机数。
timestamp	时间戳。本示例中使用函数now()获取当前时间戳。
signmethod	算法类型，支持hmacmd5和hmacsha1。
Sign	签名，即计算出的password。password计算方法如下：password=signHmacSha1(params, deviceConfig.deviceSecret)

设备身份认证实例代码如下：

```
var rp = require('request-promise');
const crypto = require('crypto');
const deviceConfig = {
  productKey: "替换productKey",
  deviceName: "替换deviceName",
  deviceSecret: "替换deviceSecret"
}
//1.获取身份token
rp(getAuthOptions(deviceConfig))
  .then(function(parsedBody) {
    console.log('Auth Info :',parsedBody)
  })
  .catch(function(err) {
    console.log('Auth err :'+JSON.stringify(err))
  });

//生成Auth认证的参数
function getAuthOptions(deviceConfig) {
  const params = {
    productKey: deviceConfig.productKey,
    deviceName: deviceConfig.deviceName,
    timestamp: Date.now(),
    clientId:Math.random().toString(36).substr(2),
  }
  //1.生成clientId，username，password
  var password = signHmacSha1(params,deviceConfig.deviceSecret);
  var options = {
    method: 'POST',
    uri:'https://iot-as-http.cn-shanghai.aliyuncs.com/auth',
    body: {
      "version":"default",
      "clientId":params.clientId,
      "signmethod":"hmacsha1",
      "sign": password,
      "productKey":deviceConfig.productKey,
      "deviceName":deviceConfig.deviceName,
      "timestamp":params.timestamp
    },
    json: true
  };
  return options;
}
//HmacSha1 sign
function signHmacSha1(params, deviceSecret) {
  let keys = Object.keys(params).sort();
  // 按字典序排序
  keys = keys.sort();
  const list = [];
  keys.map((key) => {
    list.push(`${key}${params[key]}`);
  });
  const contentStr = list.join('');
  return crypto.createHmac('sha1', deviceSecret).update(contentStr).digest('hex');
}
```

将以上代码中需要填入的信息替换为您的设备信息后，运行程序。认证成功，则获得token。

```
$ node device-https.js
Auth Info : { code: 0,
  info: { token: '3bf8d66a6af                    b40139' },
  message: 'success' }
```

设备认证返回的token会在一定周期后失效（目前token有效期是7天），请务必考虑token失效逻辑的处理。

2. 上报设备数据。认证通过，设备获得token后，便可使用token作为上报数据的password。设备上报数据的请求参数如下表。

参数	说明
method	请求方法。必须指定为POST。
uri	◦ endpoint地址和Topic组成uri： iot-as-http.cn-shanghai.aliyuncs.com/topic + topic。 ◦ 后一个topic需指定为设备上报属性的topic： /sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/property/post
body	设备上报的消息内容。
password	指定为设备认证返回的token。
Content-Type	设备上报的数据的编码格式。目前仅支持：application/octet-stream。

设备上报数据示例代码如下：

```
const topic = `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/property/post`;
//上报数据
pubData(topic, token, getPostData())
function pubData(topic, token, data) {
   const options = {
     method: 'POST',
     uri: 'https://iot-as-http.cn-shanghai.aliyuncs.com/topic' + topic,
     body: data,
     headers: {
       password: token,
       'Content-Type': 'application/octet-stream'
     }
   }
   rp(options)
     .then(function(parsedBody) {
       console.log('publish success :' + parsedBody)
     })
     .catch(function(err) {
       console.log('publish err ' + JSON.stringify(err))
     });
}
//模拟物模型数据
function getPostData() {
   var payloadJson = {
     id: Date.now(),
     params: {
       humidity: Math.floor((Math.random() * 20) + 60),
       temperature: Math.floor((Math.random() * 20) + 10)
     },
     method: "thing.event.property.post"
   }
   console.log("===postData\n topic=" + topic)
   console.log(payloadJson)
   return JSON.stringify(payloadJson);
}
```

运行以上代码程序后，可在本地日志中查看运行结果。

```
$ node device-https.js
Auth Info : { code: 0,
  info: { token: 'd12...                    8f41c6d' },
  message: 'success' }
===postData
 topic=/sys/a1...                    X/no3...    222/thing/event/property/post
{ id: 1564478683170,
  params: { humidity: 69, temperature: 17 },
  method: 'thing.event.property.post' }
publish success :{"code":0,"info":{"messageId":1156133486382312960},"message":"success"}
```

在物联网平台控制台上，该设备的**设备详情**页**运行状态**页签下，可查看设备上报的温湿度属性数据。

名词解释

- **设备身份认证**

  设备请求连接物联网平台时，物联网平台会根据设备上报的参数：ProductKey、DeviceName、clientId等进行身份校验。

- **token**

  设备通过HTTPS协议请求建立连接时，物联网平台根据设备上报的参数进行身份认证。认证通过后，返回设备认证token。该token将在设备上报数据请求中作为password使用。目前，token有效期是7天。

更多最佳实践

点击查看更多阿里云最佳实践。

3.设备透传数据云端解析

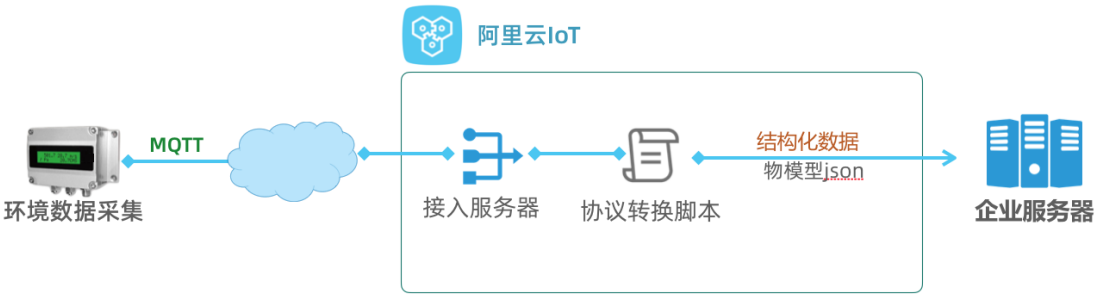
设备原始数据无需适配物模型，在物联网平台云端通过脚本实现二进制数据转换为物模型JSON格式。

背景信息

在大量物联网业务场景中，由于资源受限，或配置较低，设备端不适合直接构造物模型的JSON数据结构体与物联网平台进行直接通信。此时，可以将原数据直接透传到物联网平台，然后在物联网平台控制台，编写数据解析脚本，将设备上行数据解析为物联网平台定义的标准格式（Alink JSON）。

物联网平台接收到来自设备的数据时，先运行解析脚本，将透传的数据转换成Alink JSON格式的数据，再进行业务处理。数据解析流程图如下所示。

数据解析架构图



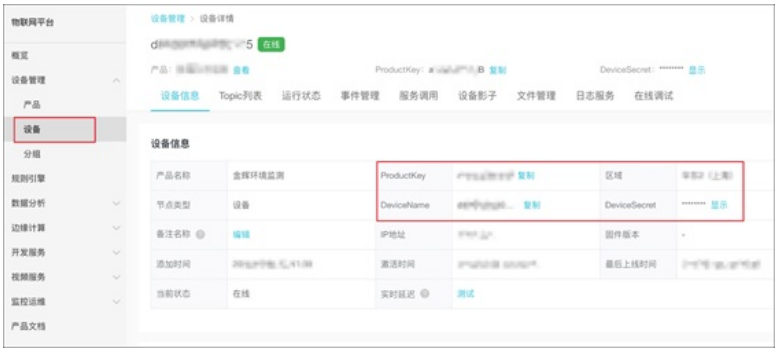
配置云端控制台

1. 创建产品并定义物模型。
   i. 根据设备实际情况，创建产品。示例中，选择**WiFi入网**，数据采用**透传/自定义**方式。

   ii. 在**功能定义**栏，自定义如下图所示功能。包括温度、湿度、二氧化碳、甲醛、PM2.5、光照强度。

2. 创建设备。

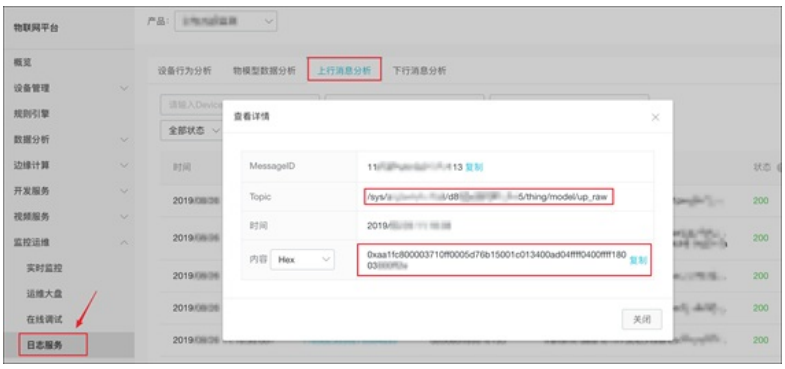
i. 在**设备**页，为该产品创建设备。示例中，使用设备序列号作为DeviceName。



ii. 记录完整的设备证书信息（ProductKey、DeviceName、DeviceSecret），以备后续设备开发使用。

数据解析

1. 查看设备上行消息。已开发完成的设备通电后，配置本地网络的WiFi账号、密码、设备身份信息后，即可在物联网平台看到设备状态为**在线**。
   i. 在物联网平台控制台左侧导航栏选择**监控运维>日志服务**，选中上面已创建的产品。
   ii. 选择上行**消息分析**页签，单击MessageID，即可查看设备上报消息详情。



我们用Hex方式查看设备上报的原始数据。

```
0xaa1fc800003710ff0005d76b15001c013400ad0ff0400ffff18003000ff2e
```

此时并不能读懂设备数据的业务意义，需要使用阿里云物联网平台云端数据解析能力。

2. 编写数据解析脚本。找到设备开发文档，获取数据协议内容。参考协议，编写解析脚本，处理设备上报的原始数据。下图为某设备数据协议示例。

消息体结构		
Byte	说明	备注
12	PM2.5 值低字节	返回：PM2.5 值(0~999ug/m3)
13	PM2.5 值高字节	
14	温度值*10 低字节	返回: 温度值（-10~50℃）
15	温度值*10 高字节	
16	湿度值低字节	返回：湿度值(0~99%)
17	湿度值高字节	
18	(二氧化碳含量)低字节	返回: 二氧化碳含量
19	(二氧化碳含量)高字节	
22	(甲醛含量*100)低字节	返回: 甲醛含量(0~9.99)
23	(甲醛含量*100)高字节	
28	照度低字节	返回:照度值(lux)
29	照度高字节	

您可以在物联网平台提供的数据解析页面，编辑、提交脚本并模拟数据解析。

数据解析脚本中需定义支持以下两个方法：

- Alink JSON格式数据转为设备自定义数据格式：protocolToRawData。
- 设备自定义数据格式转Alink JSON格式数据：rawDataToProtocol。

示例中的环境采集设备只有数据上报功能，因此只需要编写上行数据解析函数protocolToRawData，无需实现rawDataToProtocol。

```
var PROPERTY_REPORT_METHOD = 'thing.event.property.post';

//上行数据，自定义二进制转物模型JSON
function rawDataToProtocol(bytes) {
  var uint8Array = new Uint8Array(bytes.length);
  for (var i = 0; i < bytes.length; i++) {
    uint8Array[i] = bytes[i] & 0xff;
  }
  var dataView = new DataView(uint8Array.buffer, 0);
  var jsonMap = new Object();

    //属性上报method
    jsonMap['method'] =PROPERTY_REPORT_METHOD;
    //协议版本号固定字段
    jsonMap['version'] = '1.0';
    //标示该次请求id值
    jsonMap['id'] = new Date().getTime();
    var params = {};
    //12,13对应产品属性中PM2.5
    params['pm25'] =
(dataView.getUint8(13)*256+dataView.getUint8(12));
    //14,15对应产品属性中 temperature
    params['temperature'] =
(dataView.getUint8(15)*256+dataView.getUint8(14))/10;
    //16,17对应产品属性中 humidity
    params['humidity'] =
(dataView.getUint8(17)*256+dataView.getUint8(16));
    //18,19对应产品属性中co2
    params['co2'] = (dataView.getUint8(19)*256+dataView.getUint8(18));
    //22,23对应产品属性中甲醛hcho
    params['hcho'] =
(dataView.getUint8(23)*256+dataView.getUint8(22))/100;
    //28,29对应产品属性中光照lightLux
    params['lightLux'] =
(dataView.getUint8(29)*256+dataView.getUint8(28));
    jsonMap['params'] = params;
  return jsonMap;
}
//下行指令，物模型JSON转二进制格式
function protocolToRawData(json) {
  var payloadArray = [1];//此设备只有上报数据功能，无法接收云端指令
  return payloadArray;
}
```

3. 模拟数据解析。脚本编写完成后，将前面监控日志看到的设备Hex数据拷贝出来，粘贴到模拟输入框，单击**运行**，可以在右侧看到运行结果。确认数据正确后，提交脚本。

数据解析脚本提交后，进入**设备详情**页。选择**运行状态**页，显示经脚本解析后设备采集到的环境数据。

名词解释

- 物模型

  是对设备在云端的功能描述，包括设备的属性、服务和事件。物联网平台通过定义一种物的描述语言来描述物模型，称之为TSL（即 Thing Specification Language），采用JSON格式，您可以根据TSL组装上报设备的数据。

- Alink协议

  阿里云定义的设备与云端之间的通信协议。

- 设备证书

  设备证书指ProductKey、DeviceName、DeviceSecret。

  - ProductKey：是物联网平台为产品颁发的全局唯一标识。
  - DeviceName：在注册设备时，自定义的或自动生成的设备名称，具备产品维度内的唯一性。
  - DeviceSecret：物联网平台为设备颁发的设备密钥，和DeviceName成对出现。

- 数据解析脚本

  针对采用透传格式/自定义数据格式的设备，需要在云端编写数据解析脚本，将设备上报的二进制数据或自定义的JSON数据，转换为平台上的Alink JSON数据格式。

更多最佳实践

点击查看更多阿里云最佳实践。

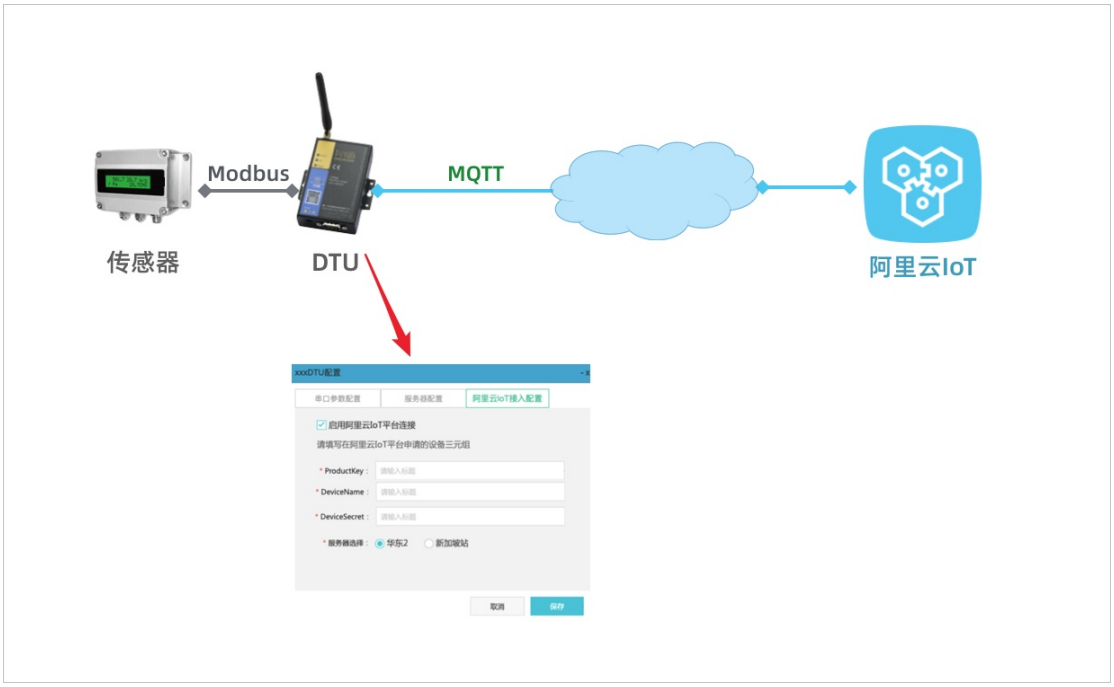
4.存量设备通过DTU上云

本文将具体介绍如何通过符合阿里云物联网平台接入协议规范的DTU设备，快速实现存量串口输出设备接入阿里云物联网平台。

背景信息

在工业、农业、医疗、城市、楼宇、园区等多种场景中，存在着大量的通过串口与外界通信的存量设备。对此类设备进行物联网改造时，往往无法修改设备本身的串口传输协议，只能在云端进行数据解析工作。为了快速使此类设备接入和使用阿里云物联网平台，阿里云联合硬件合作伙伴，共同定义了可以通过简单配置即可接入物联网平台平台的透传数据DTU设备。

数据流转流程图



本地设备通过串口与DTU设备相连，DTU通过2G、3G、4G或Ethernet网络与阿里云物联网平台相连，由DTU设备实现阿里云物联网平台的接入协议。设备证书将被配置到DTU中，由DTU代表设备与物联网平台进行数据通信。

创建产品和设备

在物联网平台创建产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

1. 登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

iii. 填入产品信息，单击**确定**。完成产品创建。存量设备的数据按其本身格式通过DTU设备透传到物联网平台，因此需创建数据格式为透传/自定义的产品。产品信息设置如下图所示。

3. 创建设备。
   i. 在控制台左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。
   iii. 选择刚创建的产品，输入设备名称和备注名称，单击**确定**。完成设备创建。

设备创建成功后，会弹出设备证书信息。您也可以在**设备管理**页，单击设备对应的**查看**按钮，进入**设备详情**页查看设备证书信息。该设备证书将被配置到DTU设备端。

定义物模型

物模型指将物理空间中的实体进行数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能（包括属性、事件、服务）。完成功能定义后，系统将自动生成该产品的物模型。本文以电机变频设备为例，需创建电机转速、电流和设置转速三个属性。

1. 在物联网平台控制台的左侧导航栏，选择**设备管理>产品**。
2. 在**产品管理**页，找到之前创建的产品，单击对应的**查看**按钮。
3. 在**产品详情**页，选择**功能定义**，再单击自定义功能对应的**添加功能**。
4. 根据下表逐个添加属性。

功能类型	功能名称	标识符	数据类型	取值范围	单位	读写类型
属性	转速	speed	int32	0 ~ 3000	rpm	只读
属性	电流	current	int32	0 ~ 30	A	只读
属性	设置转速	setspeed	Int32	0 ~ 3000	rpm	读写

编辑数据解析脚本

阿里云物联网平台支持的标准数据格式为Alink JSON格式，而存量设备的原始数据通过DTU设备透传到物联网平台，物联网平台不能直接处理此类数据。物联网平台提供数据解析功能，可将上行的自定义格式数据解析为Alink JSON格式；将下行数据解析为设备的自定义数据格式。您需在物联网平台控制台上，提交数据解析脚本供物联网平台调用。数据解析脚本需根据设备上报数据和云端下发数据进行编写。

1. 在物联网平台控制台上，变频电机产品对应的**产品详情**页，选择**数据解析**页签。
2. 在**编辑脚本**输入框中，输入解析脚本。

> **说明** 脚本代码中属性的标识符必须与定义物模型时定义的一致。

点击查看数据解析脚本编写指导。

本示例设备发送至云端的数据为16进制格式，因此脚本需将16进制格式数据格式转换为Alink JSON格式；并将云端下发的Alink JSON格式数据转换为16进制格式。本示例脚本如下：

```
var ALINK_ID = "12345";
var ALINK_VERSION = "1.1";
var ALINK_PROP_POST_METHOD = 'thing.event.property.post';
// var ALINK_EVENT_TEMPERR_METHOD = 'thing.event.TempError.post';
// var ALINK_EVENT_HUMIERR_METHOD = 'thing.event.HumiError.post';
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';
// var ALINK_SERVICE_THSET_METHOD = 'thing.service.SetTempHumiThreshold';
/* * * * * *
* 传入参数 ->
* 0102 // 共2个字节 * 输出结果 ->
*{"method":"thing.event.TempError.post","id":"12345","params":{"Temperature":
2},"version":"1.1"}
* 传入参数 ->
* 0202 // 共2个字节 * 输出结果 ->
*{"method":"thing.event.HumiError.post","id":"12345","params":{"Humidity":2},
"version":"1.1"}
*/
/*此函数用于实现设备上发数据到物模型的转换*/
function rawDataToProtocol(bytes) {
  /*将设备上报的RAW数据转换为数组其中bytes对象中存储着设备上报RAW数据*/
  var uint8Array = new Uint8Array(bytes.length);
  for (var i = 0; i < bytes.length; i++) {
    uint8Array[i] = bytes[i] & 0xff;
  }
  var params = {};                  // 定义属性存放对象
  var jsonMap = {};                 // 定义模拟Alink数据报对象
  /*填写Alink数据报协议头部分*/
  jsonMap['version'] = ALINK_VERSION;      // Alink 协议版本号
  jsonMap['id'] = ALINK_ID;                // 模拟消息ID
  jsonMap['method'] = ALINK_PROP_POST_METHOD;// 模拟设备上行数据方法：设备属性上报
  /*填写Alink数据报属性部分*/
  params['speed']= uint8Array[0];        // 将收到的第一个字节转换为转速值
  params['current'] = uint8Array[1];       // 将收到的第二个字节转换为电流
  jsonMap['params'] = params;            // 将参数打包到数据帧中
  return jsonMap;                   // 返回时会发送给IoT设备管理平台
}

//以下是部分辅助函数
```

```
function buffer_uint8(value)
{
  var uint8Array = new Uint8Array(1);
  var dv = new DataView(uint8Array.buffer,0);
  dv.setUint8(0, value);
  return [].slice.call(uint8Array);
}
function buffer_int16(value)
{
  var uint8Array = new Uint8Array(2);
  var dv = new DataView(uint8Array.buffer,0);
  dv.setInt16(0, value);
  return [].slice.call(uint8Array);
}
function buffer_int32(value)
{
  var uint8Array = new Uint8Array(4);
  var dv = new DataView(uint8Array.buffer,0);
  dv.setInt32(0, value);
  return [].slice.call(uint8Array);
}
function buffer_float32(value)
{
  var uint8Array = new Uint8Array(4);
  var dv = new DataView(uint8Array.buffer,0);
  dv.setFloat32(0, value);
  return [].slice.call(uint8Array);
}
/*此函数实现由云端下发数据转换为到设备能识别的16进制数*/
function protocolToRawData(json)
{
  var method = json['method'];
  var id = json['id'];
  var version = json['version'];
  var payloadArray = [];
  if (method == ALINK_PROP_SET_METHOD)  // 接收来自IoT设备管理平台的“设置设备属性”的命令
  {
    var send_params = json['params'];
    var prop_cur = send_params['setspeed'];  // 将设置的具体值抽取出来
     //按照自定义协议格式拼接 rawdata
    payloadArray = payloadArray.concat(buffer_uint8(0x55)); // 第一字节数据头，标识数据功能用户自定义
    payloadArray = payloadArray.concat(buffer_uint8(prop_cur)); // 第二字节，具体的设置值
  }
   return payloadArray;  // 返回时，将数据发送至设备端。
}
```

3. 测试脚本。
   - 测试上行数据解析。
     a. 选择模拟类型为**设备上报数据**。
     b. 在**模拟输入**下的输入框中，输入一个模拟数据。

        本示例脚本的逻辑为：数据的第一个字节为转速值，第二个字节为电流值。例如6410 ，64表示转速为100；10表示电流为16安培。

     c. 单击**运行**。
     d. 在右侧运行结果栏，查看解析结果。

- 测试下行数据解析。
  a. 选择模拟类型为**设备接受数据**。
  b. 在**模拟输入**下的输入框中，输入模拟下行数据。下行数据示例如下：

```
{
  "method": "thing.service.property.set",
   "id": "12345",
   "version": "1.0",
   "params": {
   "setspeed": 123
  }
}
```

  c. 单击**运行**。
  d. 在右侧运行结果栏，查看解析结果。

4. 确认脚本可用后，单击**提交**按钮，将脚本提交到物联网平台。

> **注意** 物联网平台不能调用草稿状态的脚本，只有已提交的脚本才会被调用来解析数据。

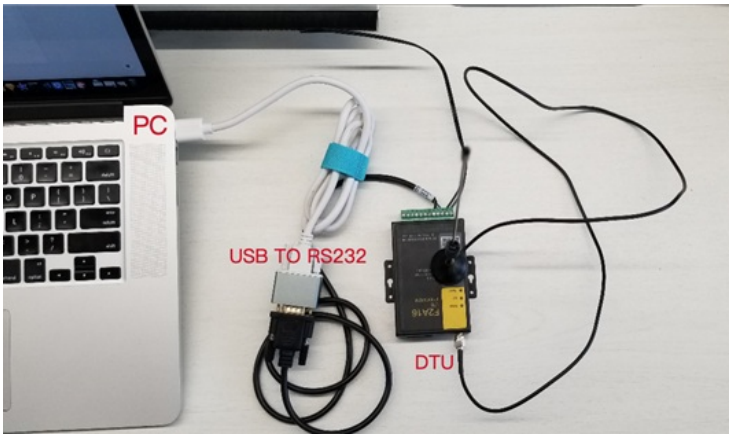
设备端开发

在本示例中，使用电脑模拟DTU设备端。电脑通过USB转串口与DTU连接。

> **注意** 请确保DTU可以正确连接Internet。

1. 配置DTU设备。本示例使用F2x16 DTU设备。

i. 连接DTU设备与电脑USB口。



ii. 在电脑上，打开DTU配置工具，配置正确的串口号，设置波特率，并打开串口。

iii. 单击右下方**登陆配置**按钮，使DTU进入配置状态。

iv. 单击**读取配置**按钮，获取现有DTU的配置。

v. 确保工作协议为port。在右侧**配置界面**下，单击**串口**，再进行本地串口配置。配置信息如下图。

vi. 单击**IoT接入配置**，填入从物联网平台获取的设备证书信息和地域。

vii. 单击下方**下发配置**按钮，使配置生效。如果配置下发失败，请单击**退出登录**后，重新配置。

viii. 完成配置后，单击**退出登录**按钮，使DTU进入正常工作模式。

ix. 将DTU断电，再重新上电。当DTU上online灯点亮后，即表示已连接上物联网平台。您还可以在物联网平台控制台上，查看设备的状态。

2. 测试数据通信。

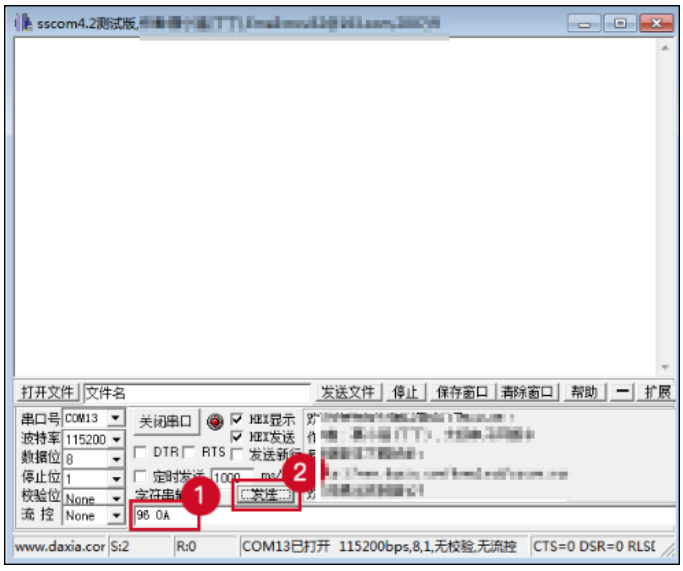
i. 测试上传数据。测试DTU设备代替存量设备上传数据到物联网平台。

a. 打开串口调试工具。

> 注意　在本地电脑上，使用串口调试工具进行设备数据的收发模拟前，请务必确保DTU配置工具已经关闭。

b. 设置串口调试工具的相关参数，并打开串口，然后单击**发送**。

根据物联网平台上的物模型定义，模拟发送转速和电流两个参数到云端。假定转速为150，电流为10安培，则在串口工具中，按先后顺序填写96 0A两个16进制数。

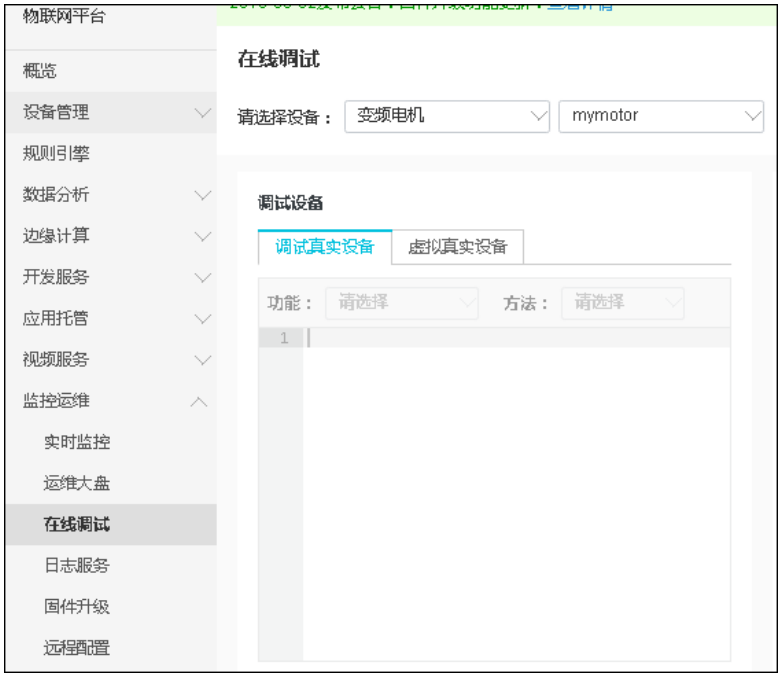


c. 数据发送后，在物联网平台控制台，设备对应的**设备详情**页**运行状态**页签下，打开**实时刷新**开关。稍后就可以看到上传的数据。

ii. 测试接收云端下发数据。使用物联网平台调试真实设备功能，下发设置转速指令，测试DTU接收云端下发数据。

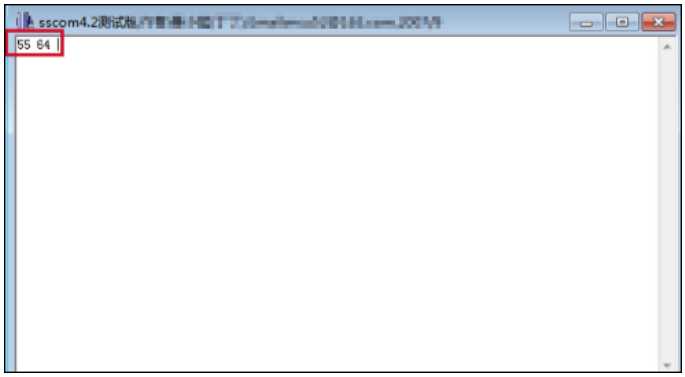
a. 在物联网平台控制台左侧导航栏，选择**监控运维>在线调试**。

b. 选择要调试的设备，再选择**调试真实设备**。



c. 选择功能为已定义的转速设置属性，方法选择为**设置**，输入一个测试值，单击**发送指令**。

d. 指令发送成功后，在DTU串口调试工具的接收框中，查看接收到的数据。接收到的数据中，55为数据头，数据值为64（即十进制的100）。



云端和设备端均能接收到正确数据，说明配置成功。

名词解释

- **物模型**

  阿里云物联网平台将设备抽象为”属性“，”事件“，”服务“三大要素。设备的三要素抽象描述，称为物模型。

- **属性**

  一般用于描述设备运行状态，例如电机的转速，灯的开关和亮度，水的温度。

- **事件**

  设备运行时的事件，一般包含需要被外部感知和处理的通知信息，例如开关机通知、报警等。

- **服务**

  设备可被外部调用的能力，例如调整电机转速。

- **设备证书**

  设备证书指ProductKey、DeviceName和DeviceSecret，是阿里云物联网平台认证设备的标识。设备证书信息不可泄露。

- **ProductKey**

  物联网平台为产品颁发的全局唯一标识。

- **DeviceName**

  在注册设备时，自定义的或系统自动生成的设备名称，具备产品维度内的唯一性。

- **DeviceSecret**

  物联网平台为设备颁发的设备密钥。

更多最佳实践

点击查看更多最佳实践。

5.博物馆环境监测LoRa设备

本示例介绍在智慧博物馆场景中，借助环境传感器设备，使用LoRa通信技术，监控藏品所在环境的温度、湿度、二氧化碳浓度、挥发有机物、甲醛、光照强度、PM2.5等环境指标。

前提条件

在进行本示例配置前，您需要完成以下准备工作：

- 注册阿里云账号，并完成实名认证。
- 使用阿里云账号开通以下阿里云产品。
  - 点击查看物联网络管理平台产品详情页。
  - 点击查看物联网平台产品详情页。
- 购买网关和环境传感器硬件。
- 购买已通过Link WAN认证的产品（内置Link WAN密钥），可访问以下地址：
  - 广域物联网。
  - 阿里云IoT元器件馆。

  收到货品后，请检查标签上的以下信息：
  - 网关设备信息：GwEUI、PIN Code。
  - 环境传感器设备信息： DevEUI、PIN Code。

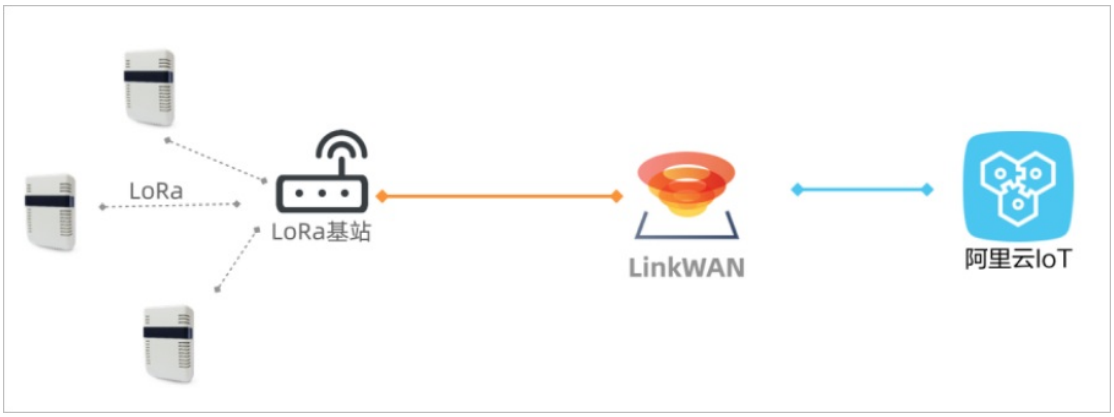
背景信息

博物馆作为藏品的收藏和展览场所，其环境直接影响藏品的保存和状态，因此实时监控各种对藏品可能产生影响的环境参数非常重要。使用LoRa环境监测设备的配置过程包括：

- 自主搭建博物馆的LoRa网络。
- 配置LoRa环境传感器入网。
- 配置环境传感器数据上报数据到物联网平台。
- 在物联网平台上提交数据解析脚本，用于解析环境传感器上报的数据。

基于LoRa自组网技术的物联网整体架构图如下。

整体架构图

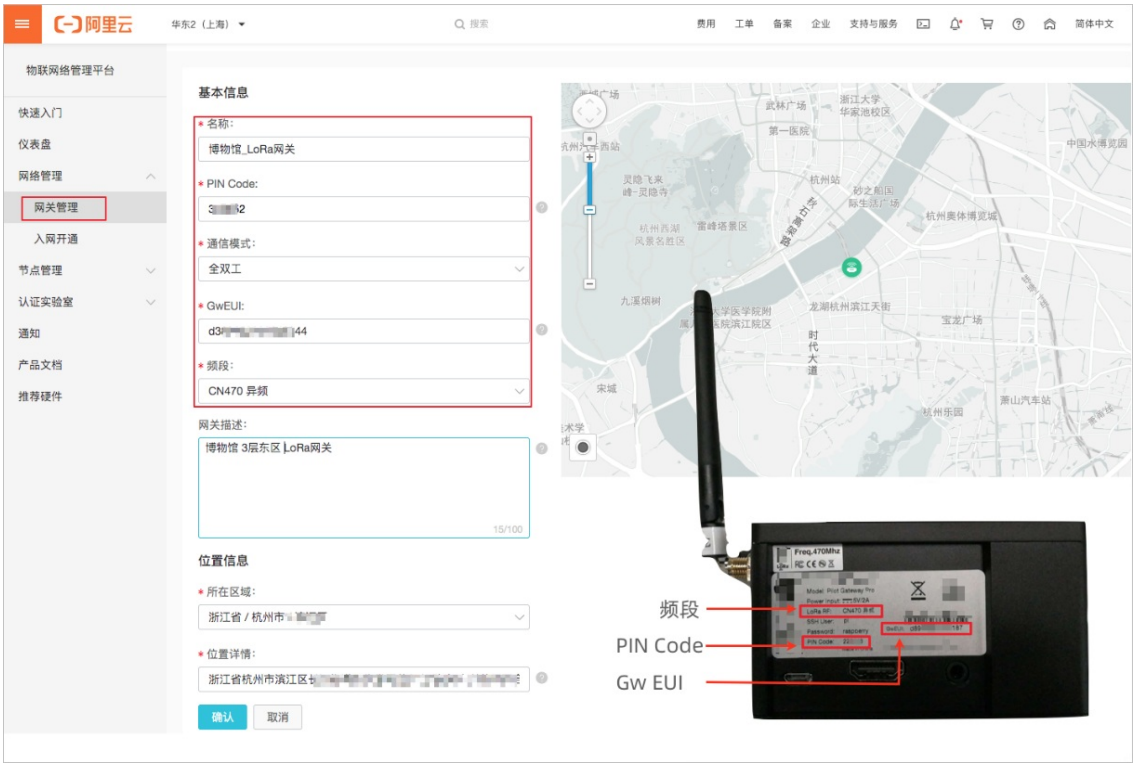


配置LoRa网关

使用LoRa设备之前，您需在物联网络管理平台上配置LoRa网关，搭建物联网所需的网络服务。

1. 登录物联网络管理平台控制台。
2. 添加网关。
   i. 在左侧导航栏，选择**网络管理>网关管理**。
   ii. 在**网管管理**页的**网管列表**页签，单击**添加网关**。

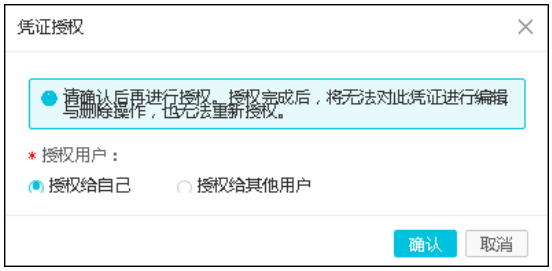
iii. 填入您的网关基本信息和位置信息后，单击**确认**。网关的GwEUI、PIN Code和频段信息，请在您的网关设备的标签上查看。如下图：



3. 添加入网凭证。
   i. 在左侧导航栏，选择**入网开通**。
   ii. 在**入网开通**页，单击**添加专用凭证**。
   iii. 填入凭证信息，单击**确认**。

4. 凭证授权。
   i. 在**入网开通**页，找到刚创建的凭证，单击凭证对应的**凭证授权**。

ii. 勾选**授权给自己**，单击**确认**。



创建产品和设备

1. 登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

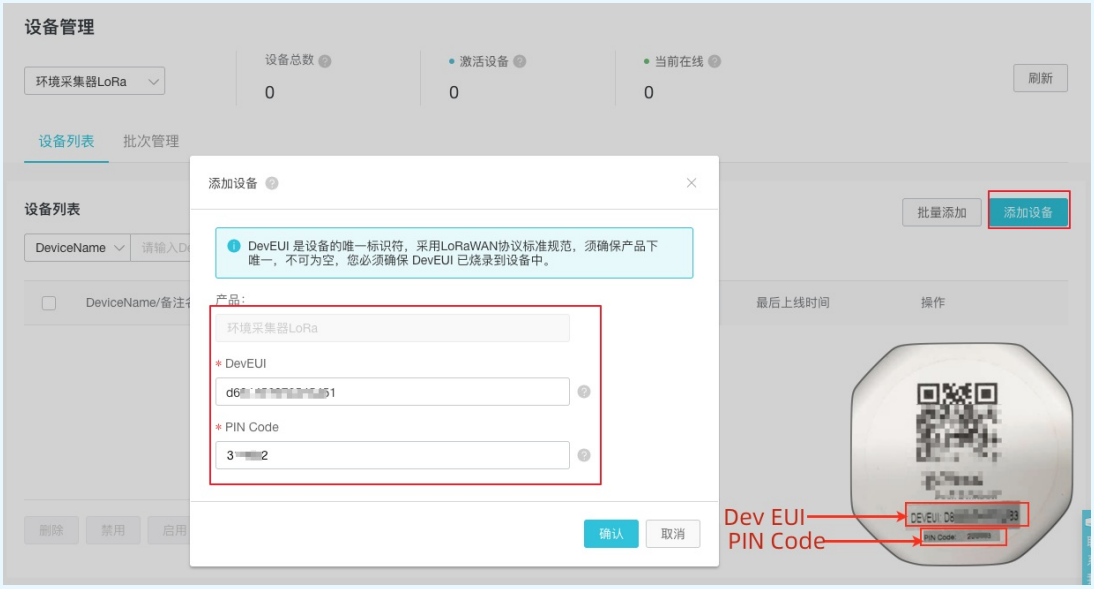
iii. 填入产品信息，单击**确定**。完成产品创建。

参数	说明
产品名称	自定义产品名称。
所属分类	选择为自定义品类。
节点类型	选择设备。
是否接入网关	选择否。
连网方式	选择为LoRaWAN。
入网凭证	选择您在物联网络平台中创建并已授权的入网凭证。
数据格式	选择为透传/自定义。

3. 创建设备。
   i. 在左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。

iii. 选择刚创建的产品，输入设备的DevEUI和PIN Code，单击**确定**。完成设备创建。

> **说明** 设备的DevEUI和PIN Code，请在您的设备标签上查看。



定义物模型

定义物模型即定义功能，包括定义属性、事件和服务。本示例中，需定义以下属性：温度、湿度、二氧化碳浓度、挥发有机物、甲醛、光照强度、PM2.5。

1. 在物联网平台控制台的左侧导航栏，选择**设备管理>产品**。
2. 在**产品管理**页，找到之前创建的产品，单击对应的**查看**按钮。
3. 在**产品详情**页，选择**功能定义**，再单击自定义功能对应的**添加功能**。
4. 逐个添加下图中的属性。

自定义功能

功能类型	功能名称	标识符	数据类型	数据定义
属性	温度	temperature	float (单精度浮点型)	取值范围：-20 ~ 70
属性	湿度	humidity	int32 (整数型)	取值范围：0 ~ 100
属性	二氧化碳	co2	int32 (整数型)	取值范围：0 ~ 10000
属性	挥发性有机物	voc	float (单精度浮点型)	取值范围：0 ~ 1000
属性	甲醛	hcho	float (单精度浮点型)	取值范围：0 ~ 5
属性	pm25	pm25	int32 (整数型)	取值范围：0 ~ 10000
属性	光照强度	lightLux	float (单精度浮点型)	取值范围：0 ~ 10000

编写数据解析脚本

本示例中，LoRa设备上报的数据是二进制格式，
如：`AA1FC800003710FF000503690B0018013500FFFFFFFFFFFFFFFFFFFFFFFFFF6C`。阿里云物联网平台的标准数据格式为Alink JSON格式，不能直接使用二进制数据进行业务处理。对此，物联网平台提供数据解析功能，您可以依据设备数据格式和物模型，编写数据解析脚本，提交至物联网平台，用于将数据解析成标准的Alink JSON。

1. 在**产品详情**页，选择**数据解析**。

2. 在**编辑脚本**输入框中，输入解析脚本。点击查看数据解析脚本编写指导。本实例的数据解析脚本如下：

```
var PROPERTY_REPORT_METHOD = 'thing.event.property.post';
/*
示例数据：
传入参数 ->
  AA1FC800003710FF000503690B0018013500FFFFFFFFFFFFFFFFFFFFFFFF6C
输出结果 ->
{
 "method":"thing.event.property.post",
 "id": 1526870753,
 "params": {
  "temperature": 10,
  "humidity": 88
 },
 "version": "1.0"
}
*/
//上行数据，自定义二进制转物模型json
function rawDataToProtocol(bytes) {
   var uint8Array = new Uint8Array(bytes.length);
  for (var i = 0; i < bytes.length; i++) {
    uint8Array[i] = bytes[i] & 0xff;
  }
  var dataView = new DataView(uint8Array.buffer, 0);
  var jsonMap = new Object();
     //属性上报method
     jsonMap['method'] = PROPERTY_REPORT_METHOD;
     //协议版本号固定字段
     jsonMap['version'] = '1.0';
     //标示该次请求id值
     jsonMap['id'] = new Date().getTime();
     var params = {};
     //12,13
     params['pm25'] = (dataView.getUint8(13)*256+dataView.getUint8(12));
     //14,15对应产品属性中 temperature
     params['temperature'] =
(dataView.getUint8(15)*256+dataView.getUint8(14))/10;
     //16,17对应产品属性中 humidity
     params['humidity'] = (dataView.getUint8(17)*256+dataView.getUint8(16));
     //18,19
     params['co2'] = (dataView.getUint8(19)*256+dataView.getUint8(18));
     //22,23
     params['hcho'] = (dataView.getUint8(23)*256+dataView.getUint8(22))/100;
     //26,27
     //params['voc'] = (dataView.getUint8(27)*256+dataView.getUint8(26))/100;
     //28,29
     params['lightLux'] = (dataView.getUint8(29)*256+dataView.getUint8(28));
     jsonMap['params'] = params;

     return jsonMap;
}
//下行指令，物模型json转二进制格式
function protocolToRawData(json) {
     var payloadArray = [];
     // 追加LoRa下行帧头部
     payloadArray = payloadArray.concat(0x5d);
     payloadArray = payloadArray.concat(0x0a);//厂商提供数据端口
     payloadArray = payloadArray.concat(0x00);
     // 追加LoRa业务内容...
     return payloadArray;
}
```

3. 测试脚本。
   i. 选择模拟类型为**设备上报数据**。
   ii. 在**模拟输入**下的输入框中，输入一个模拟数据，如：`AA1FC800003710FF000503690B0018013500FFFFFFFFFFFFFFFFFFFFFFFF6C`。
   iii. 单击**运行**。
   iv. 在右侧**运行结果**栏，查看解析结果。

4. 确认脚本能正确解析数据后，单击**提交**，将脚本提交到物联网平台系统。

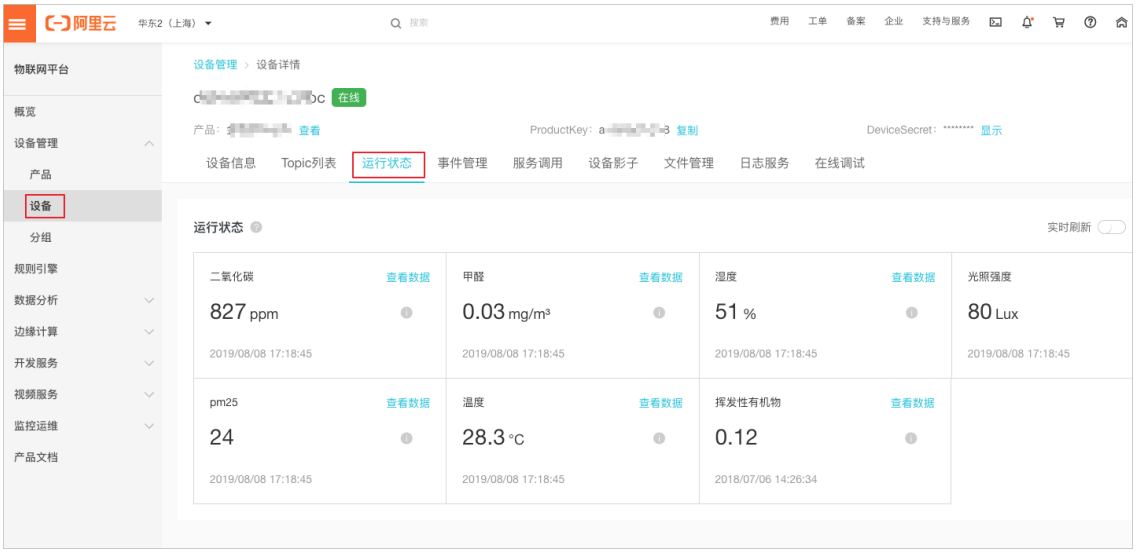
> **注意** 物联网平台不能调用草稿状态的脚本，只有已提交的脚本才会被调用来解析数据。

LoRa设备接入物联网平台

1. LoRa设备上线。将LoRa设备通电，设备便可使用LoRa网关接入物联网平台。设备接入物联网平台后，便可根据探测结果，上报环境数据。物联网平台调用您提交的数据解析脚本进行数据转换，将设备上报的二进制数据解析为Alink JSON格式。
2. 查看设备状态。在物联网平台控制台**设备管理**页，可查看设备状态。对应的设备显示为**在线**，则表示LoRa设备已接物联网平台。

3. 查看设备上报的环境参数。
   i. 在物联网平台控制台**设备管理**页，单击设备对应的**查看**查看按钮。

ii. 在**设备详情**页，选择**运行状态**。**运行状态**页签下，显示设备上报的属性数据。



说明 您还可以在物联网平台上，配置规则引擎数据流转规则，将设备上报的数据流转到其他阿里云产品中进行存储和计算处理。点击查看配置数据流转规则配置说明。

名词解释

- **物联网络管理平台（Alibaba Cloud Link WAN）**

  物联网络管理平台是阿里云面向物联网领域开发人员推出的网络管理平台，旨在帮助开发者搭建无线空口数据通道，实现终端（如传感器等）数据通过无线技术上报云端。可与阿里云物联网平台搭配使用，实现自主管理的物联网无线覆盖区。

- **物联网平台（Alibaba Cloud IoT Platform）**

  物联网平台提供安全可靠的连接通信能力，向下连接海量设备，支撑设备数据采集上云；向上提供云端API，通过API调用下发指令数据实现设备远程控制。另有设备管理、规则引擎、数据分析、边缘计算等增值服务。

- **LoRa**

  LoRa（Long Range）是基于线性扩频（CSS）的扩频调制技术，用于建立长距离、低功耗的无线局域通信链路。

- **PIN Code**

  LoRa设备的PIN码，通常印刷在设备的外显标签上。在物联网络管理平台上配置网关时，需传入网关设备的PIN Code；在物联网平台上创建设备时，需使用设备的PIN Code。

- **GwEUI**

  LoRa网关设备的唯一标识符，通常印刷在网关设备的外显标签上。在物联网络管理平台上配置网关时，需传入网关设备的GwEUI。

- **DevEUI**

  LoRa设备的唯一标识符，通常印刷在设备的外显标签上。在物联网平台上创建设备时，需使用设备的DevEUI。

更多最佳实践

点击查看更多阿里云最佳实践。

6.用IoT Studio搭建气象监测屏

本实践案例中使用LoRa气象监测设备监测气象信息，上报温度、湿度、大气压、经度、纬度等数据，并使用IoT Studio平台搭建监控大屏，展示气象监测设备最新上报的数据和历史数据曲线图。

前提条件

在进行本示例配置前，您需要完成以下准备工作：

- 注册阿里云账号，并完成实名认证。
- 使用阿里云账号开通以下阿里云产品。
  - 物联网络管理平台，点击查看产品详情页。
  - 物联网平台，点击查看产品详情页。
- 购买LoRa网关和LoRa气象监测设备硬件。

  购买已通过Link WAN认证的产品（内置Link WAN密钥），可访问以下地址：
  - 广域物联馆详情页。
  - 阿里云IoT元器件馆详情页。

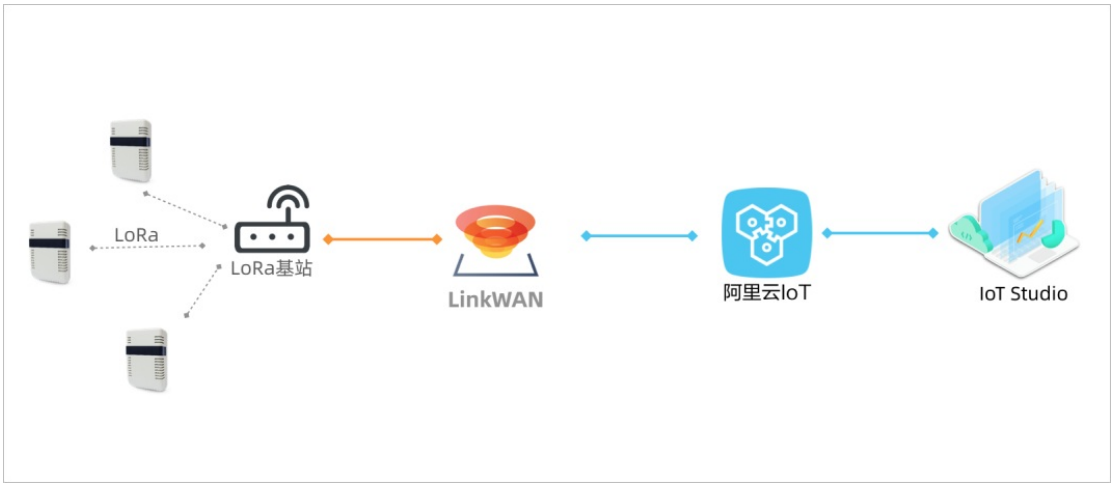
背景信息

本案例实现过程：

- 自主搭建气象站的LoRa网络。
- 配置LoRa气象监测设备接入物联网平台。
- 定义LoRa气象监测设备物模型。
- 在物联网平台上提交数据解析脚本，用于解析环境传感器上报的物模型数据。
- 在IoT Studio平台搭建监控大屏。

本案例的架构图如下。

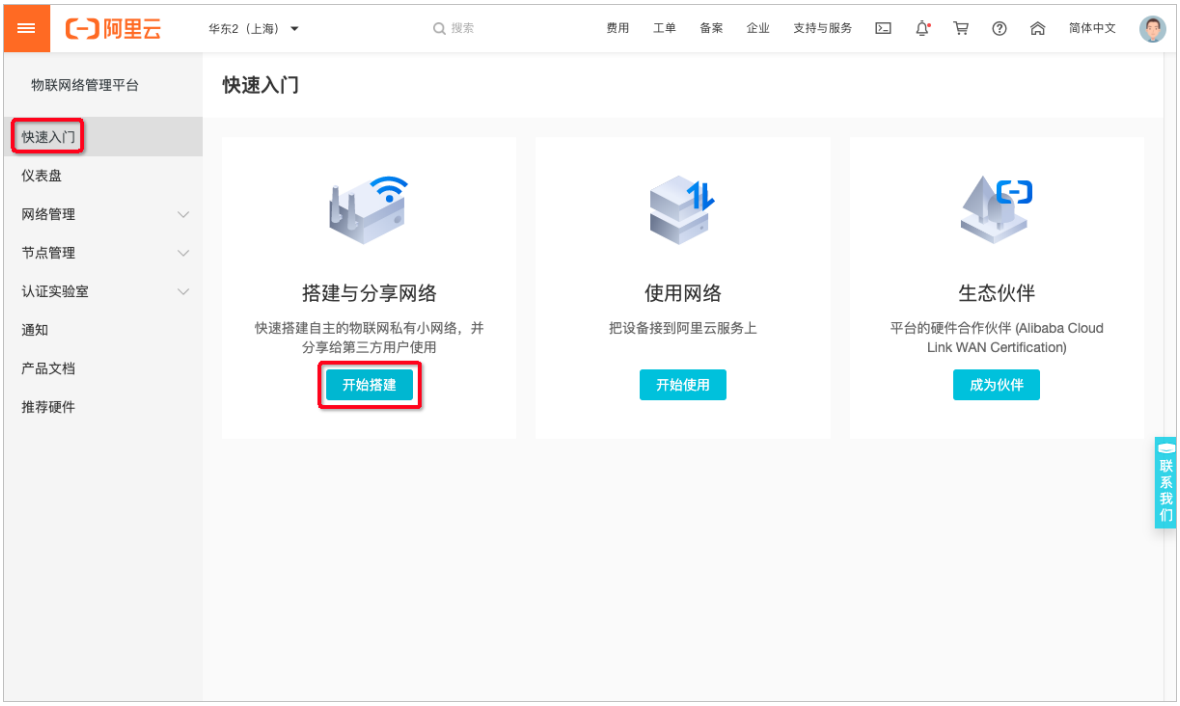
案例架构图



配置LoRa网关

使用LoRa设备之前，您需在物联网络管理平台上配置LoRa网关，搭建物联网所需的网络服务。

1. 登录物联网络管理平台控制台。
2. 在左侧导航栏，选择**快速入门**。
3. 选择**搭建与分享网络**对应的**开始搭建**。



4. 单击**开始体验**。
5. 注册网关，填入您的网关基本信息和位置信息后，单击**下一步**。网关的GwEUI、PIN Code和频段信息，请在您网关的标签上查看。如下图所示。

6. 将网关通电、连网。稍等片刻之后，网关状态显示为**在线**，则表示网关连网上线成功。

7. 添加入网凭证，单击**下一步**。

8. 将凭证授权给自己，单击**完成**。

将凭证授权给自己后，您便可以使用同一个阿里云账号，在物联网平台上使用该凭证创建连网方式为LoRaWAN的产品。

创建产品和设备

1. 登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

iii. 填入产品信息，单击**确定**。完成产品创建。

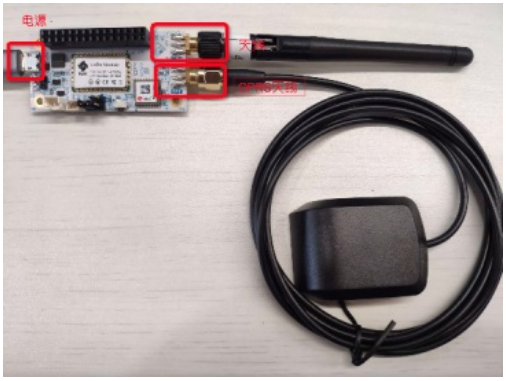
参数	说明
产品名称	自定义产品名称。
所属分类	选择为**自定义品类**。
节点类型	选择**设备**。
是否接入网关	选择**否**。
连网方式	选择为**LoRaWAN**。
入网凭证	选择您在物联网络平台中创建并已授权的入网凭证。
数据格式	选择为**透传/自定义**。

3. 创建设备。
   i. 在左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。

iii. 选择刚创建的产品，输入设备的DevEUI和PIN Code，单击**确定**。完成设备创建。

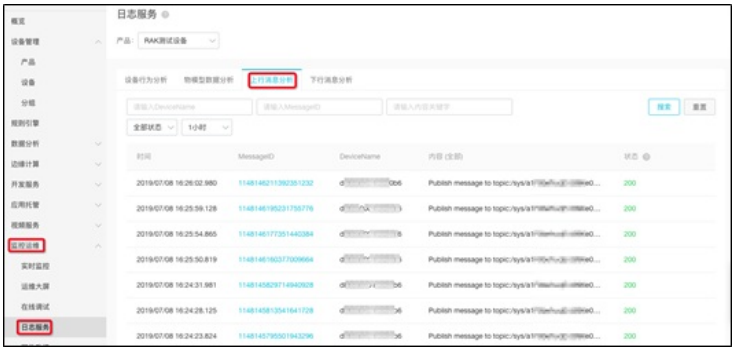
> **说明** 设备的DevEUI和PIN Code，请在您的设备标签上查看。

4. 测试设备连接物联网平台。按照设备上的标识，为设备连接天线、GPS天线、电池或电源。



设备上电约2分钟后，在物联网平台控制台**设备管理**页的**设备列表**中，该设备的状态会显示为**在线**。

在**监控运维>日志服务>上下行消息分析**页签下，可以找到该设备上报的日志。



定义物模型

物模型指将物理空间中的实体进行数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能（包括属性、事件、服务）。完成功能定义后，系统将自动生成该产品的物模型。本示例中，气象监测设备上报温度、湿度、气压、地理位置坐标等信息。因此，先在物联网平台上，为这些信息定义数据模型，即定义对应的属性。

1. 在物联网平台控制台的左侧导航栏，选择**设备管理>产品**。
2. 在**产品管理**页，找到之前创建的产品，单击对应的**查看**按钮。

3. 在**产品详情**页，选择**功能定义**，再单击自定义功能对应的**添加功能**。
4. 逐个添加下表中的属性。

> **说明** 标识符必须在产品下具有唯一性，否则数据通信时会出错。

属性名	标识符	类型	取值范围	步长	单位	读写类型
温度	Temperature	double	-99~100	0.01	℃	读写
湿度	Humidity	double	1~100	0.01	%	读写
大气压	Atmosphere	float	550 ~1060	0.01	hPa	读写
经度	Longitude	double	-180~180	0.01	°	读写
纬度	Latitude	double	-90~90	0.01	°	读写
海拔	Altitude	float	0~9999	0.01	m	读写
X加速度	Acceleration_X	float	-1000~1000	0.01	mg	读写
Y加速度	Acceleration_Y	float	-1000~1000	0.01	mg	读写
Z加速度	Acceleration_Z	float	-1000~1000	0.01	mg	读写
运行速度	Speed	float	-10000 ~10000	0.01	Km/h	读写
电池电压	Battery_voltage	float	0~100000	0.01	V	读写
气体阻力	Gas_resistance	float	-10000 ~10000	0.01	无	读写

点击查看更多新增物模型操作说明。

编写数据解析脚本

本示例中，LoRa设备上报的数据是二进制格式，如：01880537A5109D5A00846C。其中 1、2 字节为数据标识码01 88；3、4、5字节为海拔数据altitude:339m；6、7、8字节为纬度数据latitude:34.1925；9、10、11字节为经度数据longitude:108.8858。阿里云物联网平台的标准数据格式为Alink JSON格式，不能直接使用二进制数据进行业务处理；并且物联网平台下发的数据也是Alink JSON格式。物联网平台提供数据解析功能，用于解析上下行数据。您需要根据您的设备数据格式和定义的物模型，编写数据解析脚本，提交到物联网平台，以供物联网平台调用。

1. 在物联网平台控制台上，气象监测产品对应的**产品详情**页，选择**数据解析**页签。
2. 在**编辑脚本**输入框中，输入解析脚本。

> **说明** 脚本代码中属性的标识符必须与定义物模型时定义的一致。

点击查看详细的数据解析脚本编写指导。

本示例的数据解析脚本如下：

```
// var COMMAND_REPORT = 02;
// var COMMAND_SET = 01;
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //标准ALink JSON格式Topic，设备上传属性数据到云端
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';
var ALINK_VERSION = "1.1";
function rawDataToProtocol(bytes) {
  var uint8Array = new Uint8Array(bytes.length);
  for (var i = 0; i < bytes.length; i++) {
    uint8Array[i] = bytes[i] & 0xff;
  }
  var dataView = new DataView(uint8Array.buffer, 0);
  var jsonMap = {};
  // var fHead = uint8Array[0]; // 第0个BYTE为上报协议// if (fHead == COMMAND_REPORT)
  {
    jsonMap['method'] = ALINK_PROP_REPORT_METHOD; //ALink JSON格式 - 属性上报
    jsonMap['version'] = ALINK_VERSION;//ALink JSON格式 - 协议版本号固定字段
    jsonMap['id'] = '' + 12345; //ALink JSON格式 - 标示该次请求id值
    var params = {};
    switch (dataView.getInt16(0)) {
    case 0x0267:
      params['Temperature'] = Math.floor(dataView.getInt16(2) * 0.1 * 10) / 10;//保留两位小数
      params['Humidity'] = Math.floor(100 * dataView.getUint8(6) * 0.01 / 2 * 10) / 10;
      params['Atmosphere'] = Math.floor(dataView.getInt16(9) * 0.1 * 10) / 10;
      break;
    case 0x0188:
      var buffer = new Uint8Array(4);
      buffer[0] = 0;
      buffer[1] = uint8Array[2];
      buffer[2] = uint8Array[3];
      buffer[3] = uint8Array[4];
      var latitude = new DataView(buffer.buffer, 0);
      params['Latitude'] = Math.floor(latitude.getInt32(0) * 0.0001 * 10000) / 10000;
      buffer[0] = 0;
      buffer[1] = uint8Array[5];
      buffer[2] = uint8Array[6];
      buffer[3] = uint8Array[7];
      var longitude = new DataView(buffer.buffer, 0);
      params['Longitude'] = Math.floor(longitude.getInt32(0) * 0.0001 * 10000) / 10000;
      buffer[0] = 0;
      buffer[1] = uint8Array[8];
      buffer[2] = uint8Array[9];
      buffer[3] = uint8Array[10];
```

```
        var altitude = new DataView(buffer.buffer, 0);
        params['Altitude'] = Math.floor(altitude.getInt32(0) * 0.01 * 100) / 100;
        break;
      case 0x0371:
        params['Acceleration_X'] = dataView.getInt16(2);
        params['Acceleration_Y'] = dataView.getInt16(4);
        params['Acceleration_Z'] = dataView.getInt16(6);
        break;
      case 0x0702:
        params['Battery_voltage'] = dataView.getInt16(2)/10;
        params['Speed'] = Math.floor(dataView.getInt16(6) * 0.01 * 100) / 100;
        break;
      case 0x0902:
        params['Gas_resistance'] = dataView.getInt16(2);
        break;
      }
      jsonMap['params'] = params; //ALink JSON 格式 - params 标准字段 }
      return jsonMap;
    }
    function protocolToRawData(bytes) {
      var method = json['method'];
      var id = json['id'];
      var version = json['version'];
      var payloadArray = [];
      return payloadArray;
    }
}
```

3. 测试脚本。
   i. 选择模拟类型为**设备上报数据**。
   ii. 在**模拟输入**下的输入框中，输入一个模拟数据，如：`01880537A5109D5A00846C`。
   iii. 单击**运行**。
   iv. 在右侧运行结果栏，查看解析结果。

4. 确认脚本能正确解析数据后，单击**提交**，将脚本提交到物联网平台系统。

> **注意** 物联网平台不能调用草稿状态的脚本，只有已提交的脚本才会被调用来解析数据。

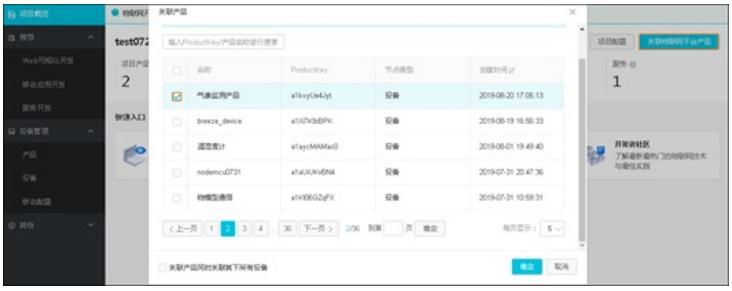
设备上报的属性数据经脚本成功解析后，将显示在该设备的**设备详情**页**运行状态**页签下。您可以在物联网平台控制台，选择**设备管理>设备>查看>运行状态**，查看属性数据。

使用IoT Studio开发监控大屏

IoT Studio平台，即物联网开发平台。您可以使用IoT Studio中的Web应用编辑器可搭建监控大屏，查看设备上报的数据。

1. 在物联网平台控制台左侧导航栏，选择**开发服务>IoT Studio**。
2. 在IoT Studio页，单击右上角**新建项目**新建项目按钮，新建一个项目。

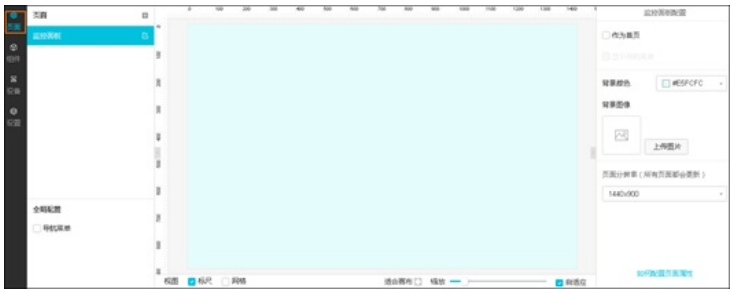
3. 在项目的**概述**或**产品**页，单击**关联物联网平台产品**，将已创建的气象监测产品与该项目关联。



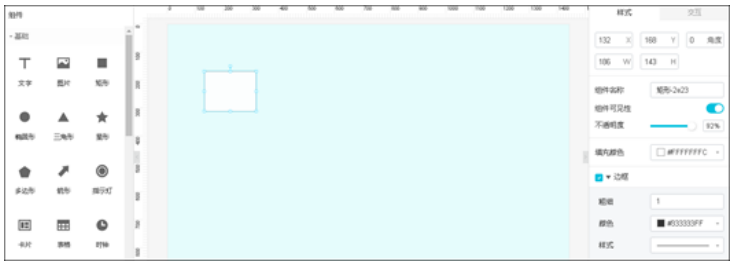
4. 在**设备**页，单击**关联物联网平台设备**，将要监控数据的来源设备与该项目关联。

5. 在左侧导航栏，选择**Web可视化开发**，然后新建一个Web应用。选择自定义模板，并输入应用名称。

6. 在Web应用编辑器中，搭建实时气象数据监控面板。
   i. 设置页面标题和背景颜色等面板页面显示效果。

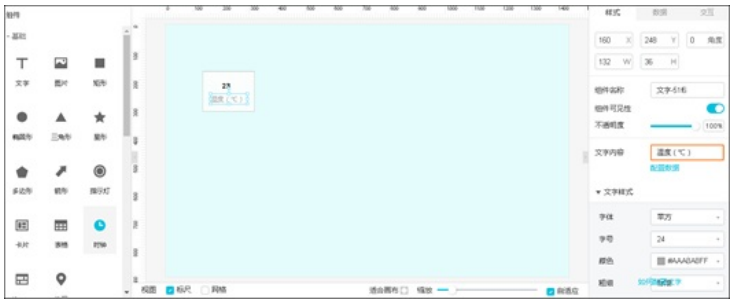


   ii. 从组件列表中，拖拽一个矩形组件到画布上，并配置组件样式。



   iii. 拖拽一个文字组件，重叠于矩形组件上，然后单击右侧配置栏中的**数据**，配置文字组件的数据源为气象监测设备的温度属性。设置完成后，该文字组件将显示气象监测设备上报的温度值。

   iv. 再拖拽一个文字组件到矩形组件上，文字内容设置为**温度（℃）**，作为温度显示组件的标题。



v. 选中刚配置好的三个组件，单击鼠标右键，选择**成组**，将这三个组件组成组件组。

vi. 根据要展示的属性数量，复制、解散组。

- 复制组

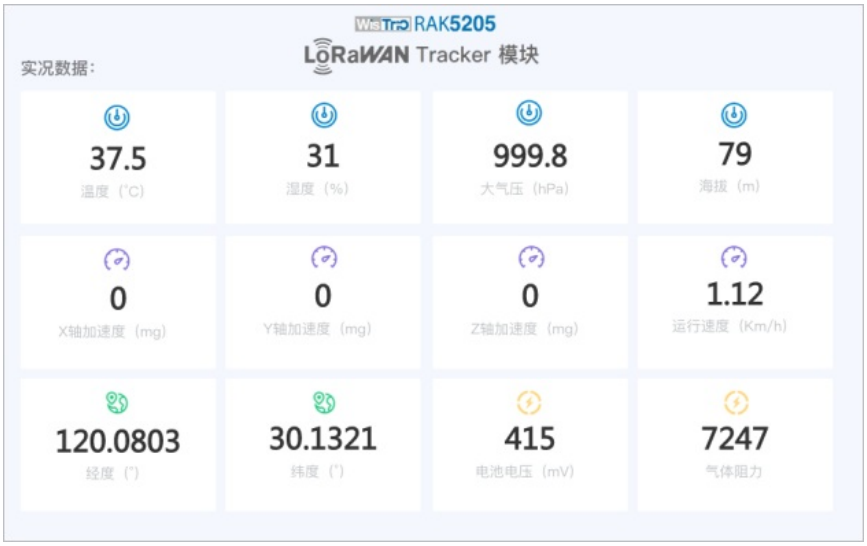
  复制组件组时，各组件的显示效果配置和数据源配置同时被复制。需配置相同显示效果的多个组件时，采用复制组件的方法，可减少配置操作。

- 解散组

  复制的组件组所有配置均相同。需先解散组，才能重新配置组件数据源等信息。

vii. 分别将数据源设置为该产品的其他属性，并设置对应的属性名称和单位。

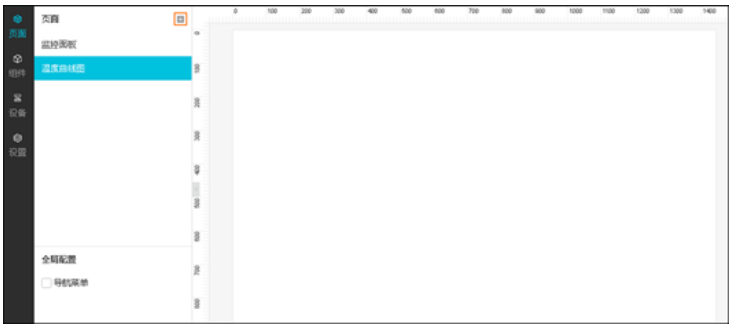
viii. 如有需要，还可在页面上增加其他组件，如图片组件等。控制面板参考图如下。



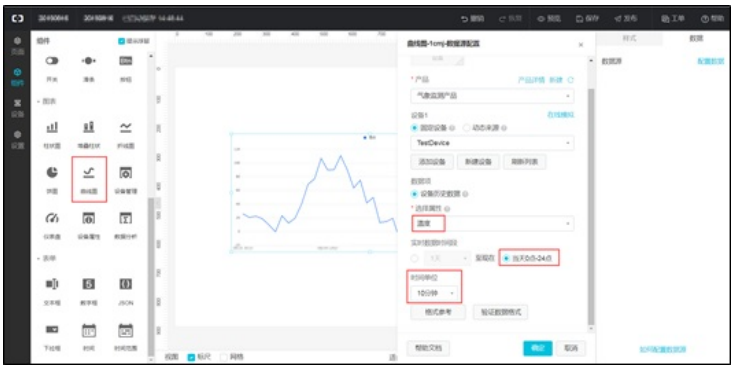
ix. 所有组件配置完成后，单击上方操作栏中的**预览**按钮，预览和测试应用页面。

7. 新建空白页面，配置历史属性数据曲线展示图。以配置温度数据展示曲线图为例。

i. 在左侧导航栏，选择**页面**，再单击新建符号“+”，新增空白页面。

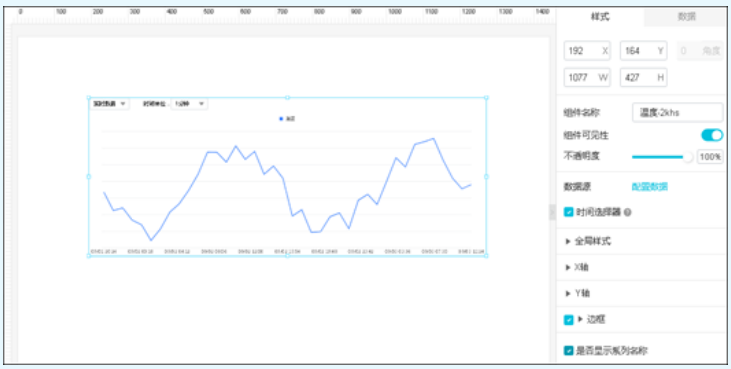


ii. 拖拽一个曲线图组件到画布上，并配置曲线组件的数据源为气象监测设备的温度属性。

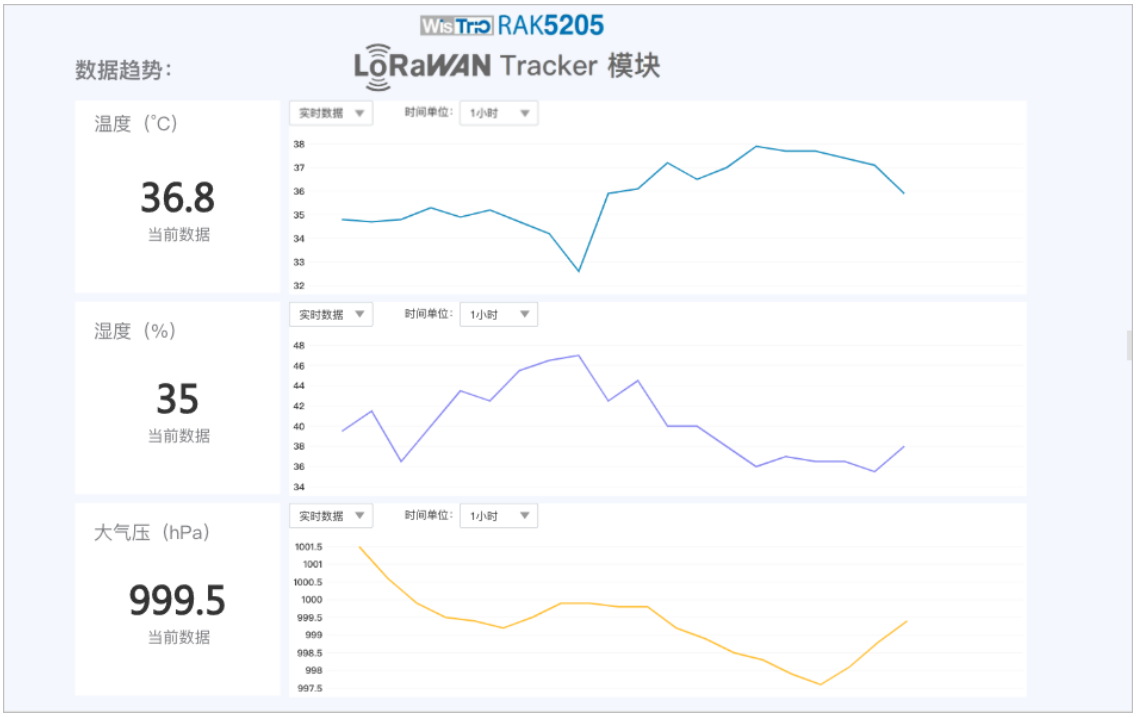


iii. 配置曲线图的显示样式。调整曲线图大小、坐标，设置是否显示时间选择器，设置系列名称为**温度**等。

> **说明** 如果勾选**时间选择器**前的复选框，表示曲线图上显示时间选择器。应用发布后，可以设置时间，查看对应的温度数据。



iv. 配置完成后，单击页面上方操作栏中的**预览**按钮，预览和测试应用页面。最终完成效果如下图所示。



8. 单击页面上方的**发布**按钮，发布应用。应用发布后，左侧导航栏中选择**设置**，可以开启应用Token验证，为应用绑定您自己的域名等。点击访问更多Web应用可视化开发操作指导。

名词解释

- **物联网络管理平台（Alibaba Cloud Link WAN）**

  物联网络管理平台是阿里云面向物联网领域开发人员推出的网络管理平台，旨在帮助开发者搭建无线空口数据通道，实现终端（如传感器等）数据通过无线技术上报云端。可与阿里云物联网平台搭配使用，实现自主管理的物联网无线覆盖区。

- **物联网平台**

  物联网平台提供安全可靠的连接通信能力，向下连接海量设备，支撑设备数据采集上云；向上提供云端API，通过API调用下发指令数据实现设备远程控制。另有设备管理、规则引擎、数据分析、边缘计算等增值服务。

- **IoT Studio**

  IoT Studio是阿里云针对物联网场景提供的应用开发工具，提供了移动可视化开发、Web可视化开发、服务开发与设备开发等一系列便捷的物联网开发工具，解决物联网开发领域开发链路长、技术栈复杂、协同成本高、方案移植困难的问题。

- **LoRa**

  LoRa（Long Range）是基于线性扩频（CSS）的扩频调制技术，用于建立长距离、低功耗的无线局域通信链路。

- **PIN Code**

  LoRa设备的PIN码，通常印刷在设备的外显标签上。在物联网络管理平台上配置网关时，需传入网关设备的PIN Code；在物联网平台上创建设备时，需使用设备的PIN Code

- **GwEUI**

  LoRa网关设备的唯一标识符，通常印刷在网关设备的外显标签上。在物联网络管理平台上配置网关时，需传入网关设备的GwEUI。

- **DevEUI**

  LoRa设备的唯一标识符，通常印刷在设备的外显标签上。在物联网平台上创建设备时，需使用设备的DevEUI。

更多最佳实践

点击查看更多阿里云最佳实践。

7.RTOS设备通过TCP模组上云

运行RTOS系统的设备通过AT指令驱动通信模组接入物联网平台，实现数据采集上报。

前提条件

在进行本示例配置前，您需要完成以下准备工作：

- 开通物联网平台。
- 注册阿里云账号，并完成实名认证。点击查看物联网平台产品详情页。
- 准备MCU、通信模组开发板和软件开发环境。

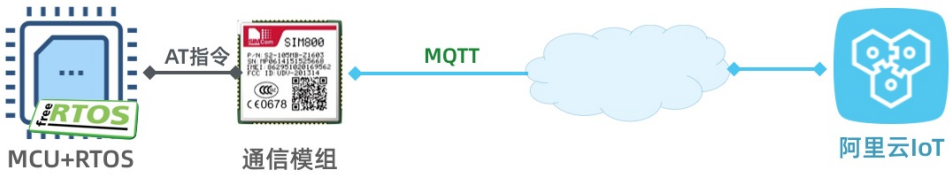
  本示例中使用的硬件和软件：

  - MCU为ST公司生产的STM32F103。点击查看详情。
  - 开发板NUCLEO-F103RB。点击查看详情。
  - 通信模组为SIMCom公司（芯讯通无线科技有限公司）生产的SIM800C。点击查看详情。
  - 开发板SIM800C mini V2.0。点击查看淘宝购买链接。
  - 开发环境为IAR Embedded Workbench for ARM。点击查看详情。

背景信息

有大量的工业自动化设备、数据采集设备、实时控制设备、家电等，原来使用的是搭载实时操作系统（RTOS）的微控制单元（MCU）。在对此类设备进行物联网改造时，可以使用阿里云物联网平台提供的C语言设备端SDK。将MCU与通信模组相连，MCU与通信模组间通过AT指令进行连接和通信。在通信模组上，使用C语言设备端SDK实现与物联网平台的连接和通信。

整体架构图



创建产品和设备

在物联网平台注册产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需配置到设备端SDK中。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

1. 登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。

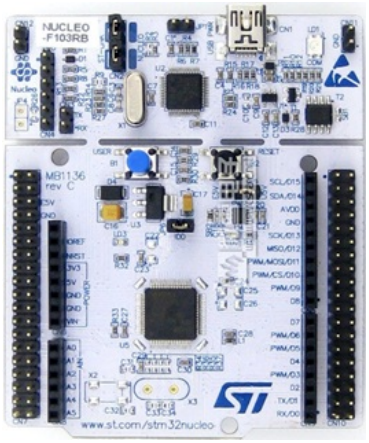
iii. 填入产品信息，单击**确定**。完成产品创建。

3. 创建设备。
   i. 在左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。
   iii. 选择刚创建的产品，输入设备名称和备注名称，单击**确定**。完成设备创建。设备创建成功后，会弹出设备证书信息。您也可以在**设备管理**页，单击设备对应的**查看**按钮，进入**设备详情**页查看设备证书信息。

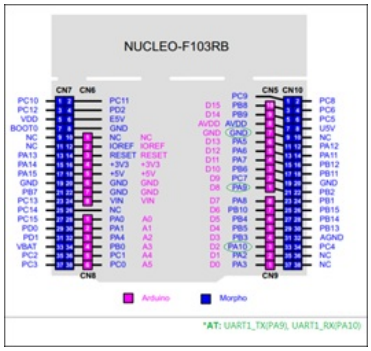
开发设备端

将MCU与通信模组开发板相连，搭建软件开发环境，创建工程项目，导入SDK，完成SDK配置。本示例中使用了以下两个开发板：

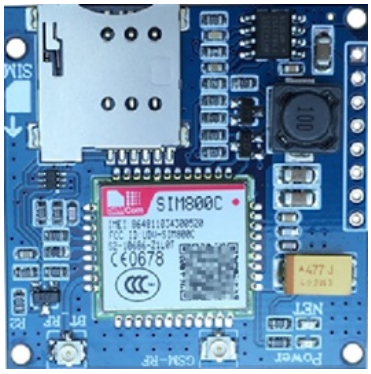
- 开发板NUCLEO-F103RB。



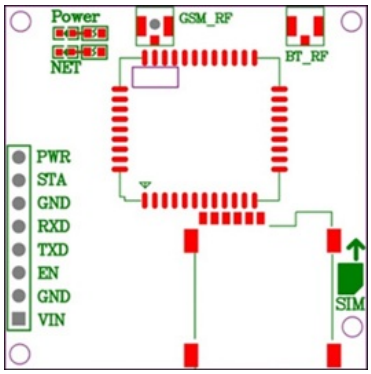
引脚示意图如下。



- MCU是SIM800C mini v2.0。

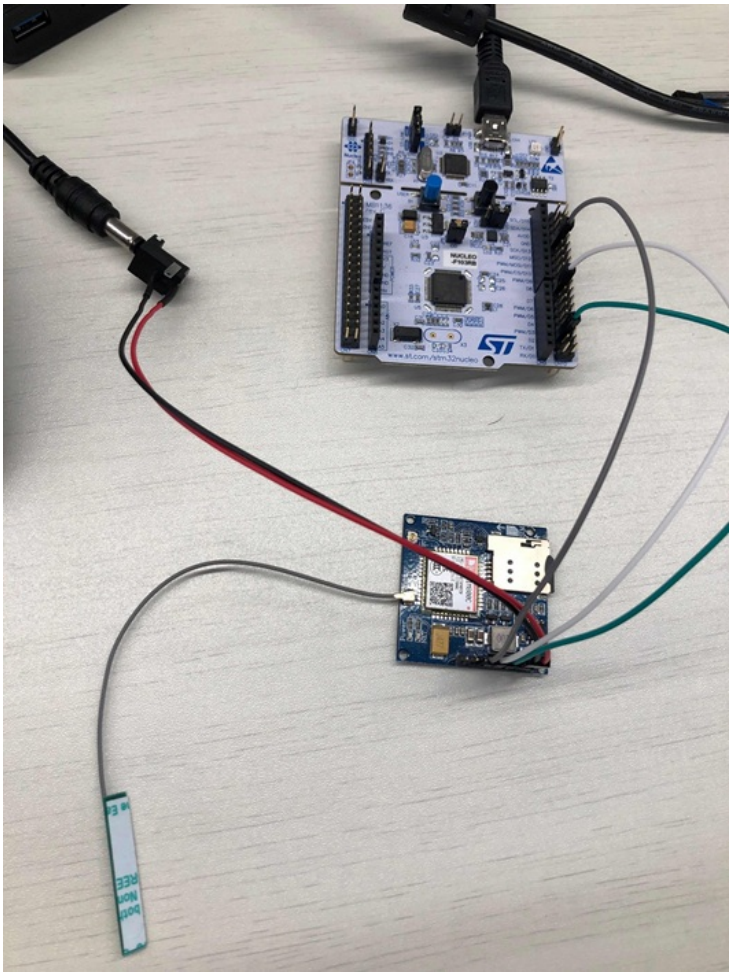


引脚示意图和说明如下。

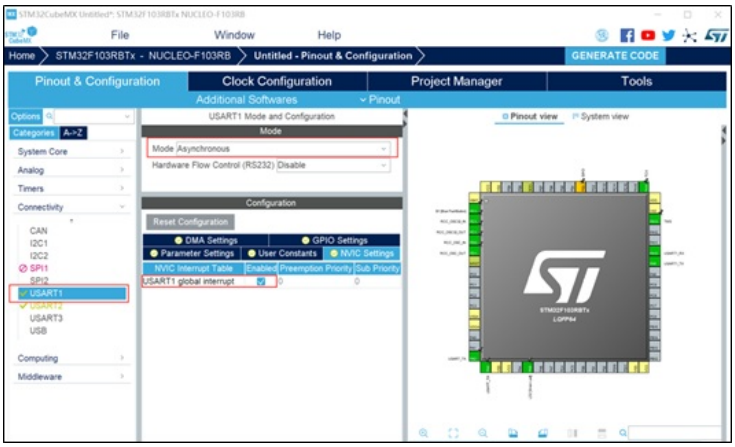


引脚	说明
PWR	开关机引脚。默认为自动开机。
STA	状态监测引脚。
GND	电源接地引脚。
RXD	接收串口引脚。
TXD	发送串口引脚。
EN	电源使能引脚。
VIN	5~18V电源输入。

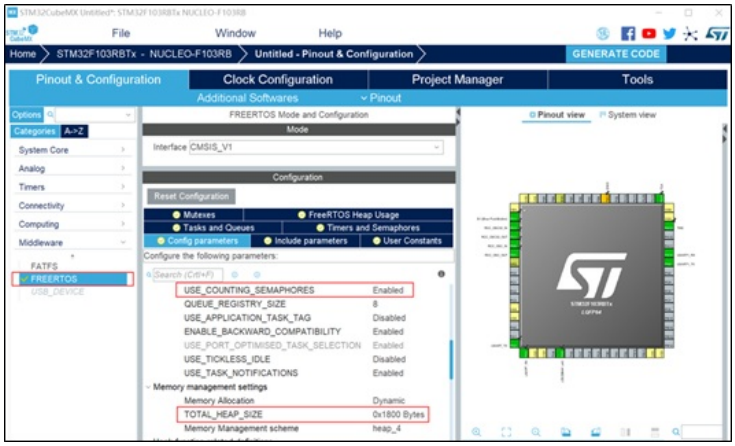
1. 连接硬件。将两个开发板的接收和发送串口连接，作为AT指令通道，如下图所示。



2. 搭建软件环境。
   i. 打开ST-cubemx，新建Project。点击查看ST-cubemx使用详情。
   ii. 在**Board Seletor**中，选择**NUCLEO-F103RB开发板**。
   iii. 在**Connectivity**菜单中，添加串口**USART1**作为MCU与模组通信的端口，并进行以下配置。
      - 在**ConnectivityUSARTx**。
      - 将**USART1**(AT端口)的**interrupt**设置为**enable**。



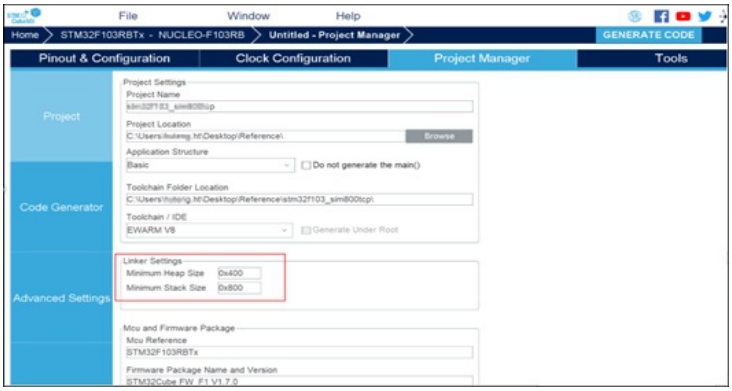
iv. 在**Middleware**菜单中，选择**FREERTOS**，并配置为使用计数信号量和堆大小（用于给每个线程分配栈）。



v. 单击**Apply**>**OK**。

vi. 单击**Cubemx**的**Project**选项中的**Generate Code**，并进行设置。

- **Toolchain/IDE**选择为 **IAR™**。
- **Heap/Satck size**按需进行配置。



vii. 单击**OK**生成代码工程。

3. 配置设备端SDK。本示例使用的设备端C语言SDK版本为3.0.1。

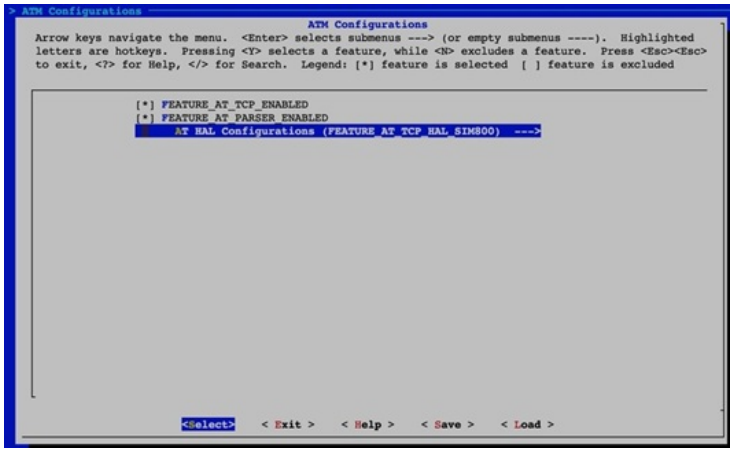
i. 点击下载SDK。

ii. 从下载包中提取SDK代码。本文以Linux系统操作为例。

a. 运行make menuconfig。

b. 选中ATM Configurations，单击Select。

c. 选中AT HAL Configurations，单击Select。



d. 配置如下项目。

```
FEATURE_PLATFORM_HAS_STDINT=y
FEATURE_PLATFORM_HAS_OS=y
FEATURE_INFRA_STRING=y
FEATURE_INFRA_NET=y
FEATURE_INFRA_LIST=y
FEATURE_INFRA_LOG=y
FEATURE_INFRA_LOG_ALL_MUTED=y
FEATURE_INFRA_LOG_MUTE_FLW=y
FEATURE_INFRA_LOG_MUTE_DBG=y
FEATURE_INFRA_LOG_MUTE_INF=y
FEATURE_INFRA_LOG_MUTE_WRN=y
FEATURE_INFRA_LOG_MUTE_ERR=y
FEATURE_INFRA_LOG_MUTE_CRT=y
FEATURE_INFRA_TIMER=y
FEATURE_INFRA_SHA256=y
FEATURE_INFRA_REPORT=y
FEATURE_INFRA_COMPAT=y
FEATURE_DEV_SIGN=y
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_MQTT_DEFAULT_IMPL=y
FEATURE_MQTT_DIRECT=y
FEATURE_DEVICE_MODEL_CLASSIC=y
FEATURE_ATM_ENABLED=y
FEATURE_AT_TCP_ENABLED=y
FEATURE_AT_PARSER_ENABLED=y
FEATURE_AT_TCP_HAL_SIM800=y
```

e. 配置完成后，在Linux中运行 `./extract.sh` 提取代码。

```
huteng@r10c05067:~/c-sdk$ ./extract.sh
. Download request sent, waiting respond ...
. Respond generating, wait longer
. Retried 1/20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  107k  100  107k    0     0   408k      0 --:--:-- --:--:-- --:--:--  408k

Please pick up extracted source files in [/disk1/huteng/c-sdk/output]
```

提取的代码位于*output/eng*目录。

```
huteng@r10c05067:~/c-sdk/output/eng$ ls
atm  dev_sign  infra  mqtt  sdk_include.h  wrappers
```

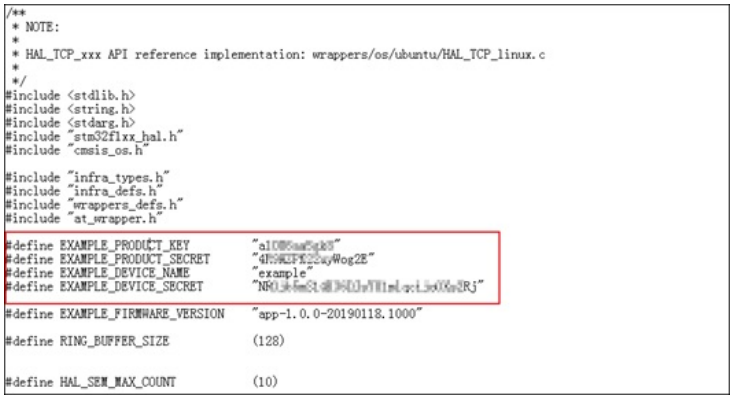
其中，各子目录分别包含的代码如下表。

目录	代码内容
atm	AT指令收发模块
dev_sign	设备身份认证模块
infra	内部实现模块
mqtt	MQTT协议模块
wrappers	HAL对接模块

iii. 在*wrappers*目录下，新建文件*wrappers.c*，该文件中的代码需实现以下HAL函数。

```
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size,
uint32_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size,
uint32_t timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void *HAL_SemaphoreCreate(void)
void HAL_SemaphoreDestroy(void *sem)
void HAL_SemaphorePost(void *sem)
int HAL_SemaphoreWait(void *sem, uint32_t timeout_ms)
int HAL_ThreadCreate(void **thread_handle,
            void *(*work_routine)(void *),
            void *arg,
            hal_os_thread_param_t *hal_os_thread_param,
            int *stack_used)
void HAL_SleepMs(uint32_t ms)
void HAL_Printf(const char *fmt, ...)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

点击下载wrappers.c文件的代码Demo。在代码Demo中，替换设备证书信息为您的设备证书信息。



```
/**
 * NOTE:
 *
 * HAL_TCP_xxx API reference implementation: wrappers/os/ubuntu/HAL_TCP_linux.c
 *
 */
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "stm32f1xx_hal.h"
#include "cmsis_os.h"

#include "infra_types.h"
#include "infra_defs.h"
#include "wrappers_defs.h"
#include "at_wrapper.h"

#define EXAMPLE_PRODUCT_KEY         "a1O[illegible]"
#define EXAMPLE_PRODUCT_SECRET      "dN9[illegible]"
#define EXAMPLE_DEVICE_NAME         "example"
#define EXAMPLE_DEVICE_SECRET       "NH[illegible]"

#define EXAMPLE_FIRMWARE_VERSION    "app-1.0.0-20190118.1000"

#define RING_BUFFER_SIZE            (128)


#define HAL_SEM_MAX_COUNT           (10)
```

说明 如果您不是使用NUCLEO-F103RB通信模组开发板，需在配置时设置：

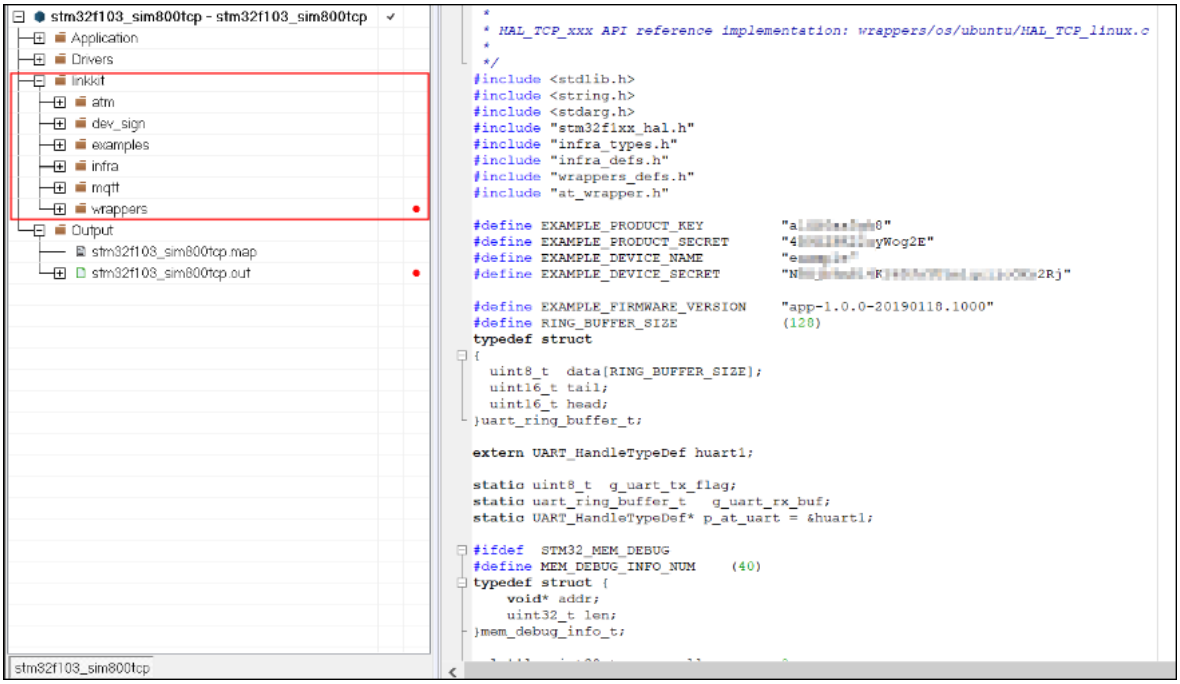
```
FEATURE_AT_TCP_HAL_SIM800=n
```

*wrappers.c*文件中的代码需实现以下HAL函数。

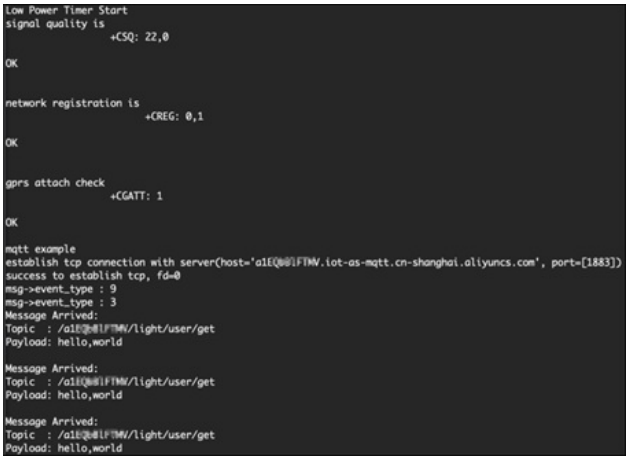
```
int HAL_AT_CONN_Close(int fd, int32_t remote_port)
int HAL_AT_CONN_Deinit(void)
int HAL_AT_CONN_DomainToIp(char *domain, char ip[16])
int HAL_AT_CONN_Init(void)
int HAL_AT_CONN_Send(int fd, uint8_t *data, uint32_t len, char remote_ip[16],
int32_t remote_port, int32_t timeout)
int HAL_AT_CONN_Start(at_conn_t *conn)
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size,
uint32_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size,
uint32_t timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)void
HAL_MutexUnlock(void *mutex)
void *HAL_SemaphoreCreate(void)
void HAL_SemaphoreDestroy(void *sem)
void HAL_SemaphorePost(void *sem)
int HAL_SemaphoreWait(void *sem, uint32_t timeout_ms)
int HAL_ThreadCreate(void **thread_handle,
            void *(*work_routine)(void *),
            void *arg,
            hal_os_thread_param_t *hal_os_thread_param,
            int *stack_used)
void HAL_SleepMs(uint32_t ms)
void HAL_Printf(const char *fmt, ...)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

整合SDK

1. 将SDK整合到IAR工程。如下图所示。



2. 运行SDK。运行成功后，设备端本地日志如下图。



在物联网平台控制台，**监控运维>日志服务**中，也可以查看设备上报数据到云端的日志。

名词解释

- **MCU**

  微控制单元(Microcontroller Unit)，又称单片机。

- **RTOS**

  实时操作系统（Real Time Operating System），用于快速接收和处理数据，实时完成任务。

- **设备端SDK**

  阿里云物联网平台提供的Link Kit SDK即设备端SDK，用于设备端开发。设备需要支持TCP/IP协议栈才能集成Link Kit SDK。设备厂商将设备端SDK集成到设备上，设备便可通过该SDK安全地接入到阿里云物联网平台。目前，物联网平台提供六种设备端SDK：C SDK、Java SDK、Python SDK、Node.js SDK、Android SDK和iOS SDK。本示例使用的是C SDK。下载设备端SDK和查看使用说明书。

- **设备证书**

  设备证书指ProductKey、DeviceName和DeviceSecret，是阿里云物联网平台认证设备的标识。设备证书信息不可泄露。

- **ProductKey**

  物联网平台为产品颁发的全局唯一标识。

- **DeviceName**

  在注册设备时，自定义的或系统自动生成的设备名称，具备产品维度内的唯一性。

- **DeviceSecret**

  物联网平台为设备颁发的设备密钥。

更多最佳实践

点击查看更多阿里云最佳实践。

8.Modbus设备通过边缘网关上云

无需编程，通过边缘计算网关把工业领域常见的Modbus协议的硬件设备接入IoT云平台。

前提条件

根据本产品的环境要求，选择运行边缘计算产品的硬件载体，例如准备一个硬件网关或者PC机，然后安装本产品。

- 环境要求

  三个版本的产品对环境的要求如下。

产品版本	硬件CPU架构	硬件CPU主频	硬件RAM	硬件磁盘
专业版	x86-64	≥2 GHZ	≥2 GB	≥2 GB
标准版	○ x86-64 ○ ARMv8-64 ○ ARMv7 VFPv3硬浮点型 ○ ARMv7软浮点型	≥1 GHZ	≥128 MB	≥128 MB
轻量版	○ x86-64 ○ ARMv8-64 ○ ARMv7 VFPv3硬浮点型 ○ ARMv7软浮点型	不限制	≥1 MB	≥1 MB

- 环境设置

  我们以在x86_64 Ubuntu 16.04机器上安装Link IoT Edge标准版产品为例，设置Link IoT Edge运行所依赖的环境。

  i. 在x86_64 Ubuntu 16.04机器的本地终端窗口或者SSH终端窗口执行以下命令，下载环境检查工具并运行。

  ```
  wget http://iotedge-web.oss-cn-shanghai.aliyuncs.com/public/testingTool/link-iot-edge_env-check.sh
  sudo chmod +x./link-iot-edge_env-check.sh
  sudo ./link-iot-edge_env-check.sh
  ```

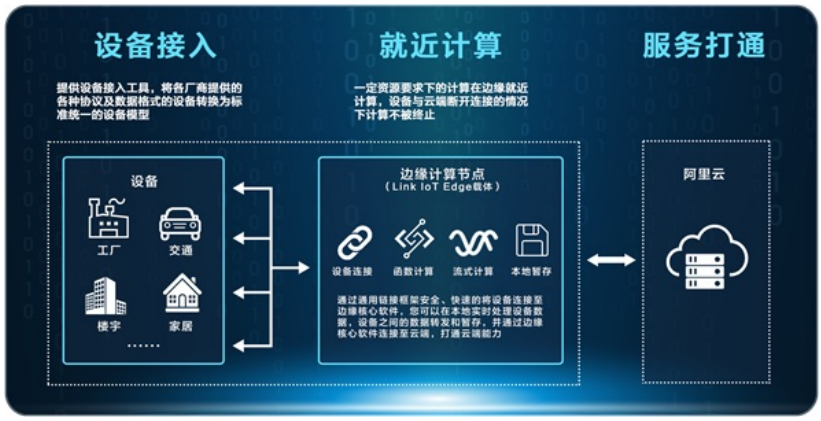
  > **说明** link-iot-edge_env-check.sh 脚本在Link IoT Edge支持的平台需要以root权限运行并需要以下Linux系统命令：printf、echo、cat、ls、expr、grep、test、uname、set、head、sort、cut、uniq、xargs、ifconfig。

  ii. 按照运行环境检查工具的提示在您的机器上安装所有必需的依赖项，当检查工具成功运行完成后，返回如下图信息，表示Link IoT Edge能够在您的机器上成功运行。

```
Link IoT Edge v1.8 Standard Run Environment required
Linux Kernel Version:
for x86_64:     >= 2.6.32
for ARMv7:      >= 3.2.0
for ARMv8_64:   >= 3.7.0

RAM:            >= 128MB
FLASH:          >= 128MB
CPU Frequency: >= 1GHZ
CPU Architecture: x86_64, ARMv7, ARMv7 VFPv3, ARMv8_64

Link IoT Edge v1.8 Run Environment Check
Check Depended Commands:
[  yes  ]  Checking for command wget
[  yes  ]  Checking for command realpath
[  yes  ]  Checking for command tar
[  yes  ]  Checking for command unzip
[  yes  ]  Checking for command readlink
[  yes  ]  Checking for command basename
[  yes  ]  Checking for command dirname
[  yes  ]  Checking for command pidof
[  yes  ]  Checking for command df
[  yes  ]  Checking for command grep
[  yes  ]  Checking for command ps
[  yes  ]  Checking for command kill
[  yes  ]  Checking for command dirname
[  yes  ]  Checking for command xargs
[  yes  ]  Checking for command umount
[  yes  ]  Checking for command unshare
[  yes  ]  Checking for command awk
[  yes  ]  Checking for command mkdir

System configuration:
Kernel architecture:   x86_64
Kernel version:   4.15
Total memory:   7.7g
Available flash space:   17g
C library version:   2.27
C library:   Ubuntu GLIBC 2.27-3ubuntu1
Init process:   /lib/systemd/systemd
Loopback is:   up

Link IoT Edge v1.8 Run Environment Check Result
Suggestion:

1. It looks like the kernel uses 'systemd' as the init process.
Please using systemd to manage Link IoT Edge service.

You can install the Link IoT Edge  v1.8 software on the device.
```

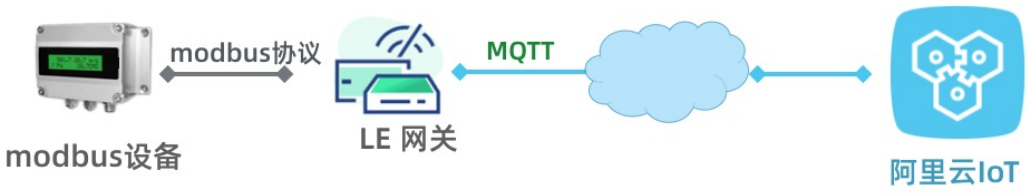
背景信息

Modbus协议是一种较为通用的数据传输协议，覆盖大量的物联网设备场景，因其通信报文种类多，报文中数据解析方式更加灵活而广泛应用于当今工业电子设备。然而，不同的场景需要将报文转换类型和报文解析类型进行组合，因此对Modbus协议的通用性提出了非常高的要求。阿里云提供的一款面向边缘场景的云边一体的PaaS层软件服务：物联网边缘计算产品（Link IoT Edge），很好地解决了Modbus协议通用性问题。该产品以官方驱动（Modbus官方驱动）的方式提供了Modbus协议设备接入能力，该驱动目前支持8种标准功能码，同时支持多达11种数据解析方式，能够满足大多数物联网场景的使用。物联网边缘计算架构图如下。



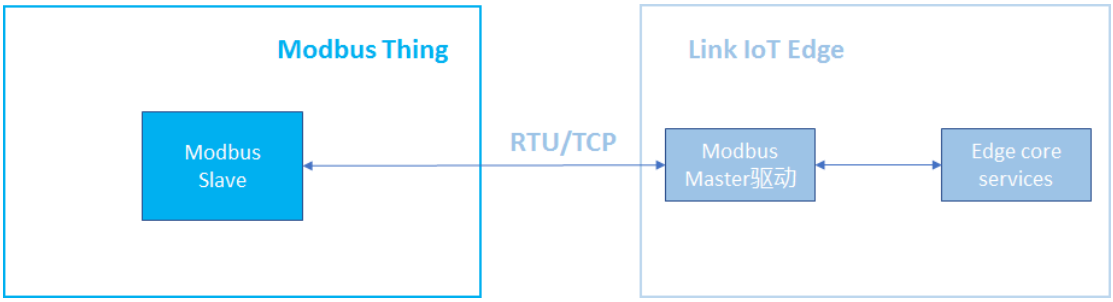
使用物联网边缘计算，您可以将云端的能力通过适当剪裁和兼容性适配映射到边缘，将云端的服务下沉到边缘，解决边缘实时性、可靠性、运维经济性等方面的问题。在设备接入侧，Link IoT Edge提供了通信协议框架方便软硬件开发者便捷开发；在云端提供了Open API帮助SaaS开发者快速构建云端应用；同时也提供了一体化的运维工具，方便您在云端集中运维，降低运维成本，提升运维效率。设备接入本地边缘计算网关，网关通过Internet接入物联网平台。整体技术架构如下所示。

整体技术架构图



阿里云物联网边缘计算产品的官方Modbus驱动示意图如下所示。网关通过本地连接方式（例如通过网络或串口）连接边缘设备，并通过边缘设备中的驱动能力，采集设备数据。

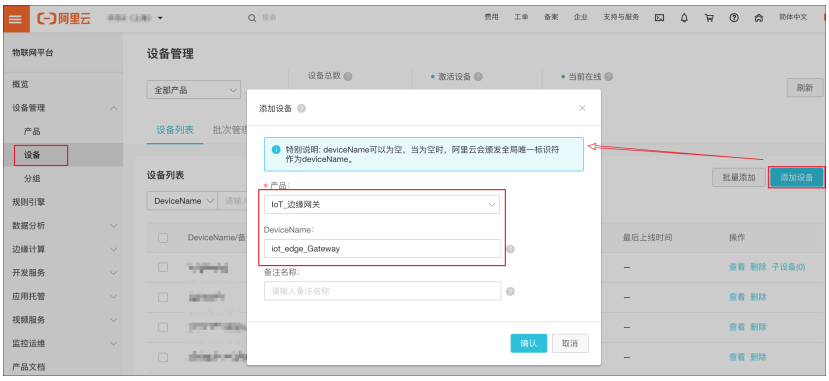
官方Modbus驱动示意图



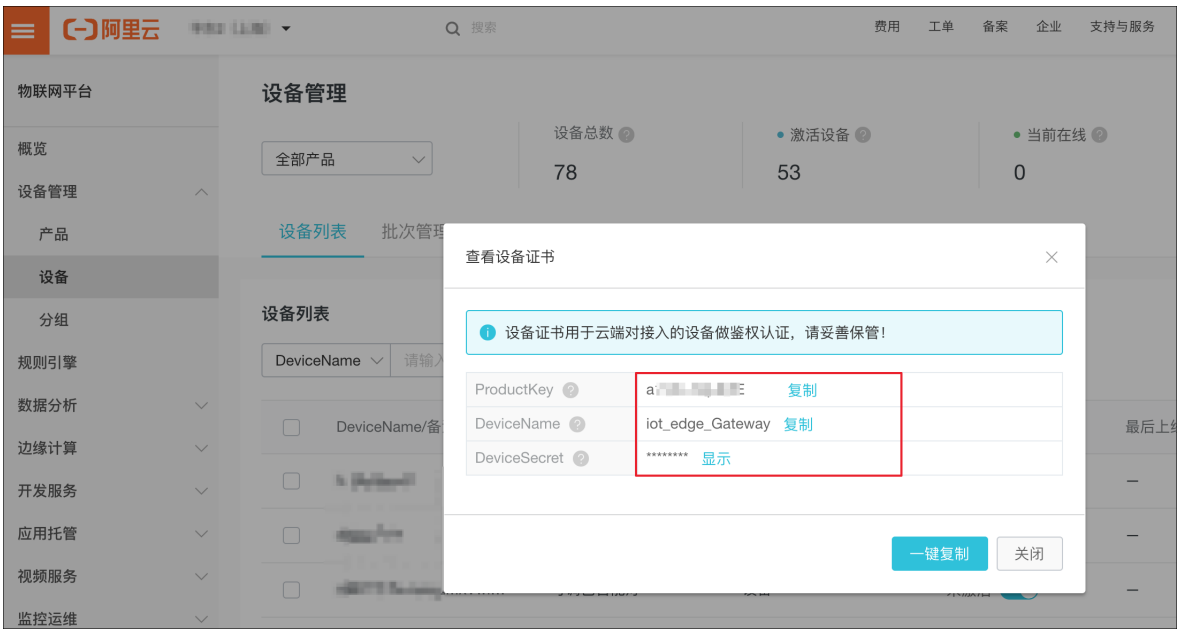
创建边缘计算网关

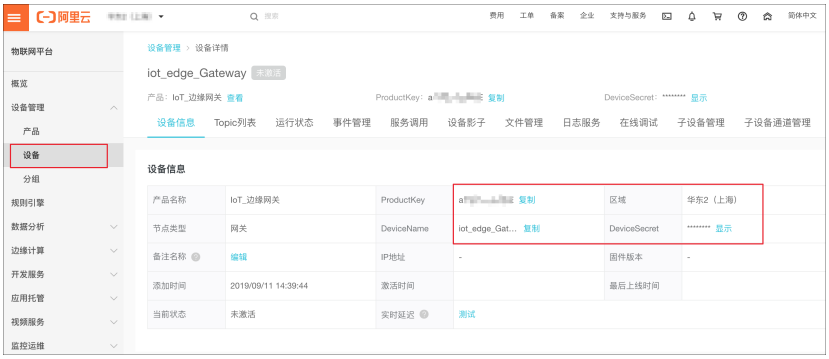
1. 登录物联网平台控制台，左侧导航栏选择**设备管理>产品**，单击**创建产品**。
2. 按照下图内容填写产品信息，然后单击**完成**。

3. 左侧导航栏选择**设备管理>设备**，在页面右侧单击**添加设备**。为上一步创建的网关产品添加一个具体的网关设备。



网关设备创建成功，可获取到设备证书信息，用于后续步骤中网关连接物联网平台的身份验证。





至此，已完成了边缘计算网关的创建。

创建Modbus传感器

接下来要在物联网平台，为用于采集环境数据的Modbus协议六合一传感器设备，创建产品模型。

1. 登录物联网平台控制台，左侧导航栏选择**设备管理>产品**，单击**创建产品**。
2. 根据实际产品情况，填写如下图信息。

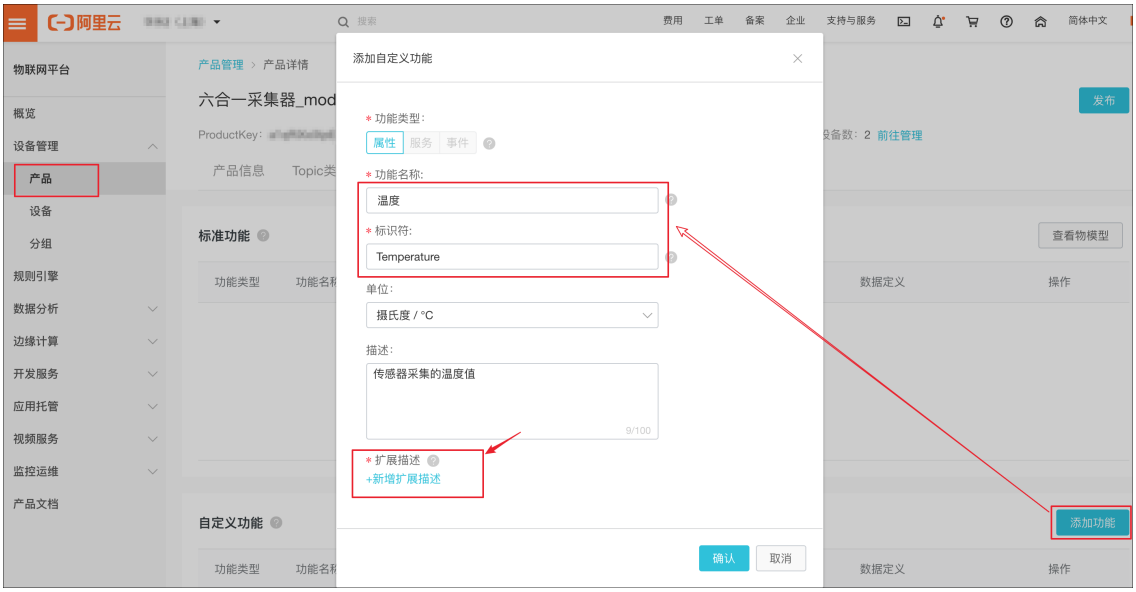
其中，部分参数说明如下。

参数	描述
节点类型	选择**设备**
是否入网关	选择**是**
接入网关协议	选择Modbus

创建成功后，可以在**产品详情**页面确认关键信息是否正确。

3. 在**产品详情**页面**功能定义**页签，为产品添加物模型（添加自定义功能）。本文以添加温度属性为例，演示物模型属性创建过程。

   i. 根据下图内容，添加属性，填写功能名称、标识符。



ii. 查看六合一传感器的说明书，找到温湿度相关的文档。示例如下：

读取设备地址 0x01 的温湿度值：

问询帧

地址码	功能码	起始地址	数据长度	校验码低位	校验码高位
0x01	0x03	0x00 0x00	0x00,0x02	0xC4	0x0B

应答帧

地址码	功能码	有效字节数	湿度值	温度值	校验码低位	校验码高位
0x01	0x03	0x04	0x02 0x92	0xFF 0x9B	0x5A	0x3D

温度：当温度低于 0℃时以补码形式上传

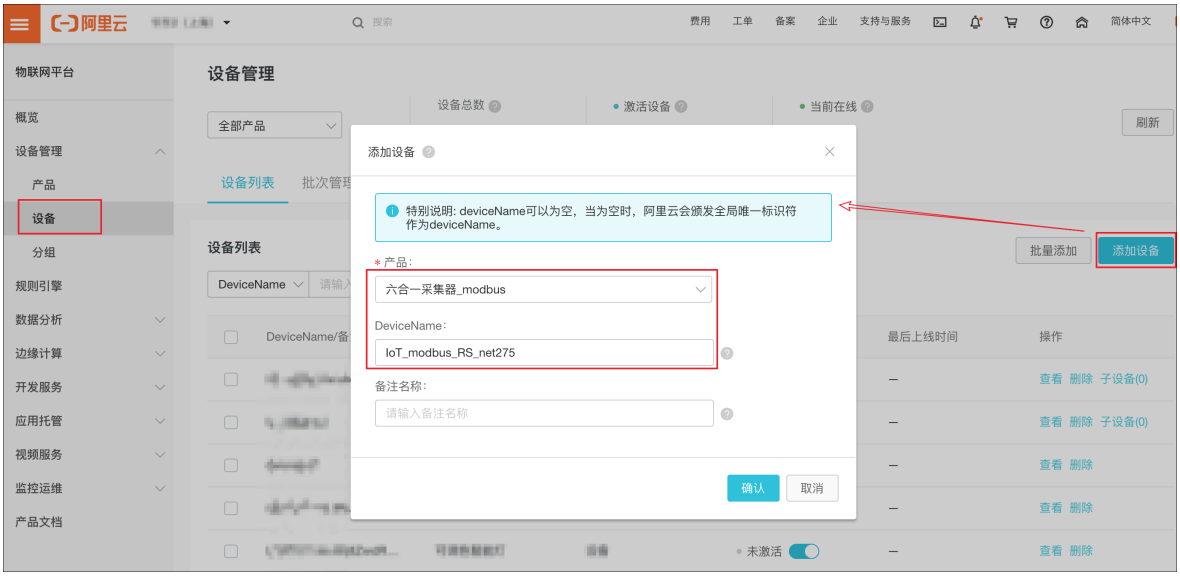
FF98 H(十六进制)=-101=>温度=-10.1℃

通过阅读文档，可以了解到温度获值转换的方法。

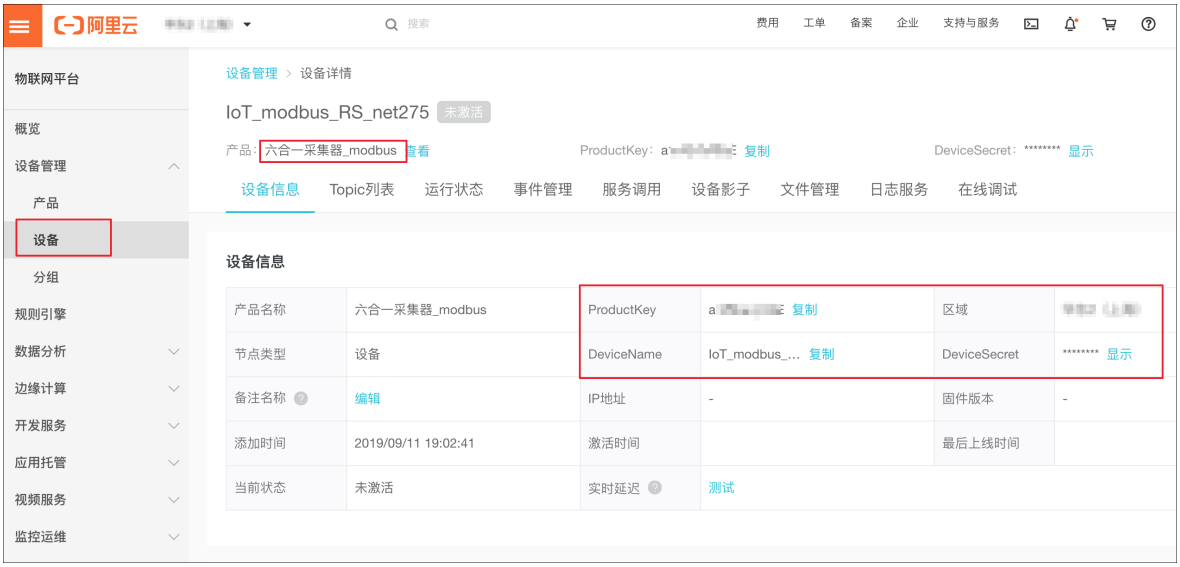
iii. 单击功能定义下面的**扩展描述**，填写**操作类型**、**寄存器地址**、**数据类型**，以及**缩放因子**，然后单击**保存**，参考下图。

至此，完成了温度属性的定义。同样方式，为产品添加湿度、气压、二氧化碳、PM2.5、光照强度属性定义。

4. 左侧导航栏选择**设备管理>设备**，在页面右侧单击**添加设备**，基于产品注册具体设备。



5. 设备注册完成后，可获取到传感器设备的证书信息。

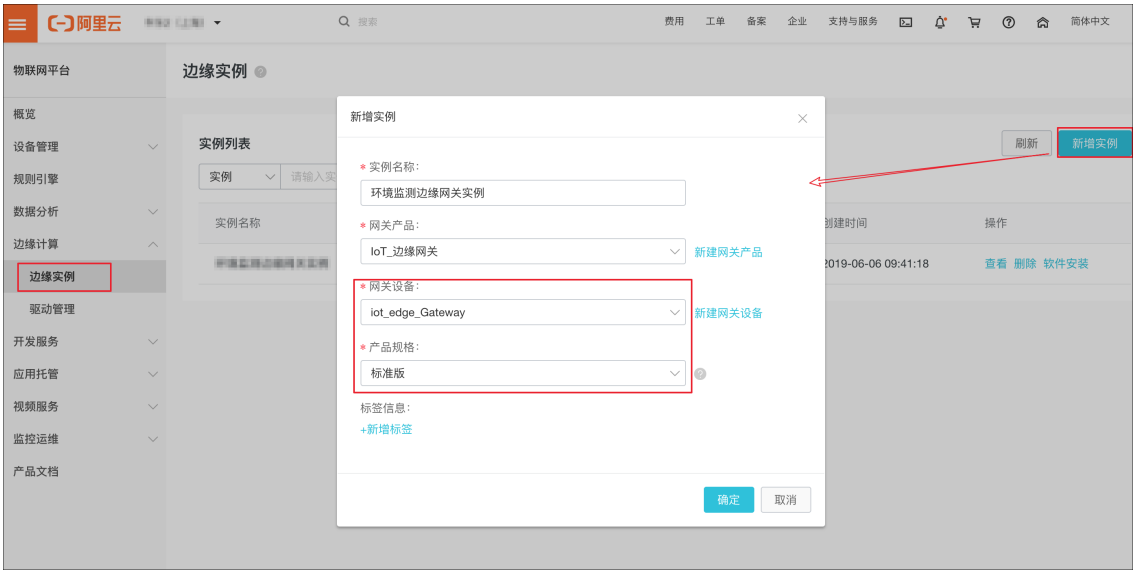


配置和部署边缘实例

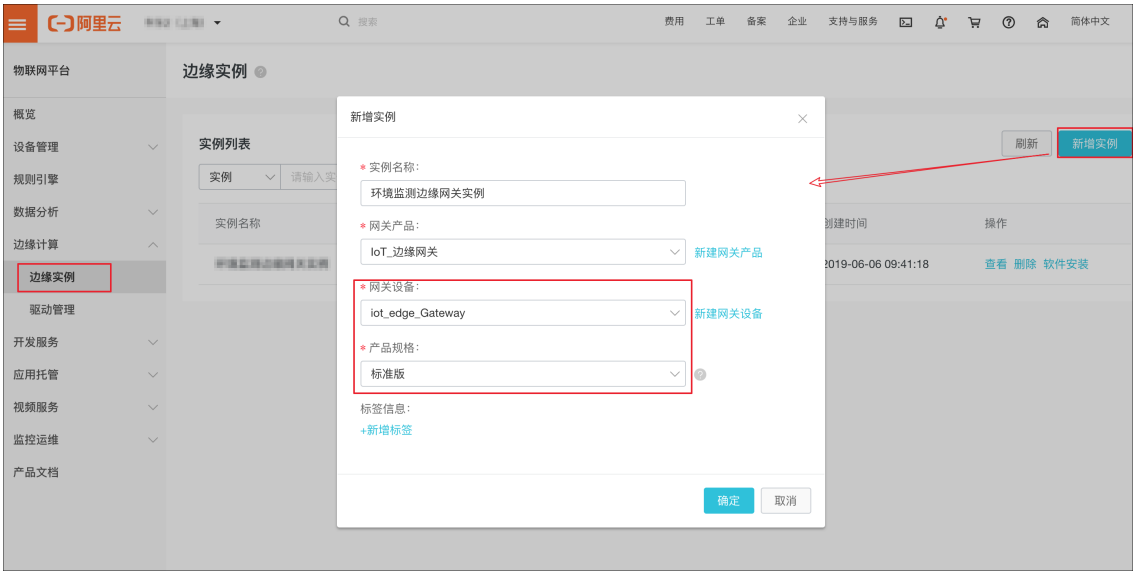
准备好边缘计算网关和Modbus采集器后，接下来就需要把二者关联起来。本文以一个边缘计算实例的方式来管理这个关系。

1. 启动网关设备。

i. 左侧导航栏选择**边缘计算>边缘实例**，单击页面右侧的**新增实例**。其中，边缘实例需要关联已创建的边缘计算网关产品和设备，如下图所示。

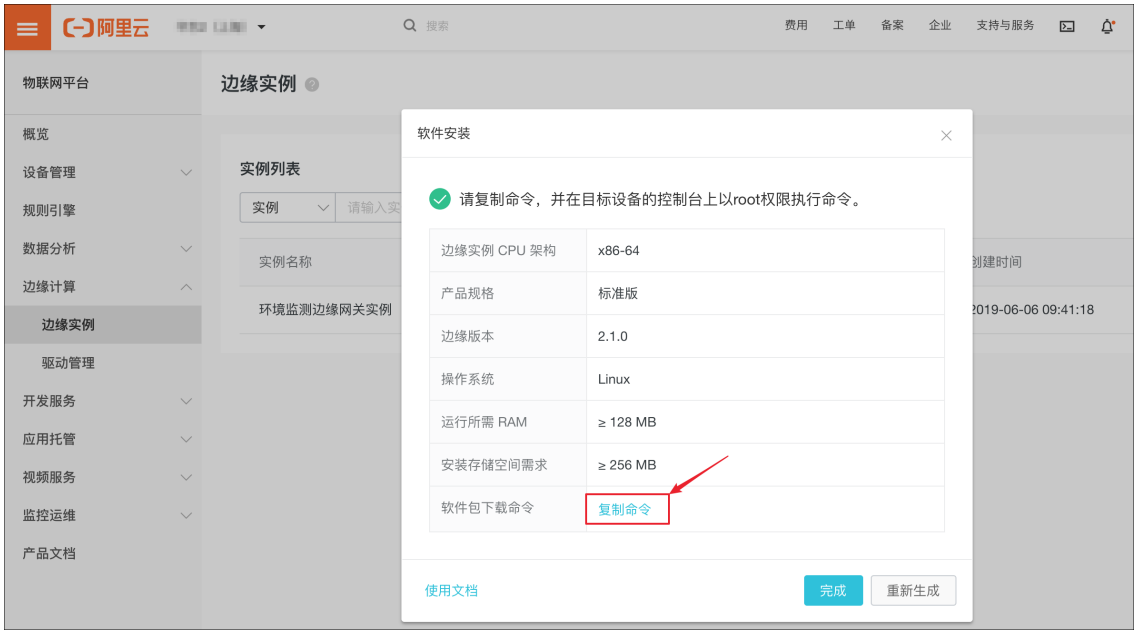


ii. 边缘实例创建完成后，需要安装边缘计算软件到网关中。单击**软件安装**，根据设备实际情况填写信息。



填好信息后，单击**生成安装命令**。

iii. 在**软件安装**话框中单击**复制命令**。



安装命令参考：

```
sudo curl -O http://link-iot-edge-packet.oss-cn-
shanghai.aliyuncs.com/config/link-iot-edge-standard.sh && sudo chmod +x link-
iot-edge-standard.sh &&./link-iot-edge-standard.sh --install x86-64 v2.1.0
&&./link-iot-edge-standard.sh --config a1*****E6E iot_edge_Gateway
d8yERLxi6faB5Kxbk8OyTJhoMK*****&& ./link-iot-edge-standard.sh --start
```

iv. 登录到网关设备，执行安装命令。该命令实现一键下载、配置并启动Link IoT Edge。命令执行完成后，会在当前目录中下载用于启动Link IoT Edge的link-iot-edge-standard.sh脚本。

v. 执行如下命令查看Link IoT Edge核心服务的运行状态。

```
sudo./link-iot-edge-standard.sh --status
```

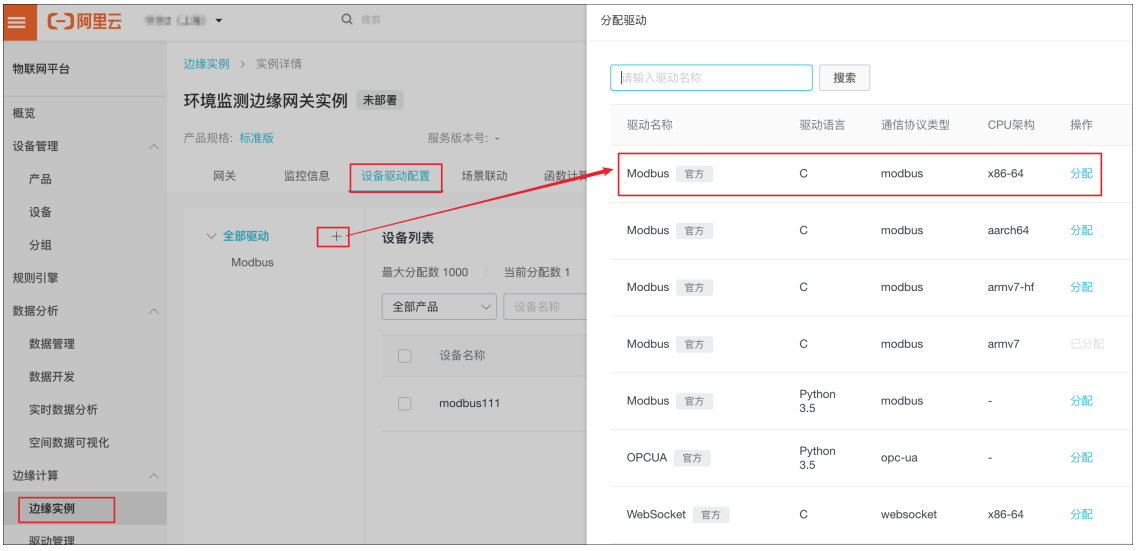
若系统显示如下信息，表示Link IoT Edge核心服务启动成功。

```
:/$ ./link-iot-edge-standard.sh --status

------------------------------------------------------------
                Link IoT Edge Service Status
------------------------------------------------------------

[  +  ]  Service mbusd.conf is active, pid is 10066
[  +  ]  Service config-manager is active, pid is 10093
[  +  ]  Service data-manager is active, pid is 10112
[  +  ]  Service watch-dog is active, pid is 10145
[  +  ]  Service cloud-proxy is active, pid is 10102
[  +  ]  Service dimu is active, pid is 10179
[  +  ]  Service message-router is active, pid is 10174
[  +  ]  Service fota is active, pid is 10195
[  +  ]  Service gateway-monitor is active, pid is 10213
[  +  ]  Service task-dispatcher is active, pid is 10208
[  +  ]  Service credential is active, pid is 10108
[  +  ]  Service fc-base is active, pid is 10359
[  +  ]  Service ifttt is active, pid is 10230
[  +  ]  Service redis-server is active, pid is 10061
[  +  ]  Service logger is active, pid is 10069
[  +  ]  Service service-monitor is active, pid is 10371
[  +  ]  Service RemoteTerminalDaemon is active, pid is 13593
[  +  ]  Service nginx is active, pid is 10043
```

说明 您也可以在物联网控制台，选择**边缘计算>边缘实例**，在已创建好的边缘实例右侧单击**查看**进入**实例详情**页面，选择**网关**，查看网关状态。

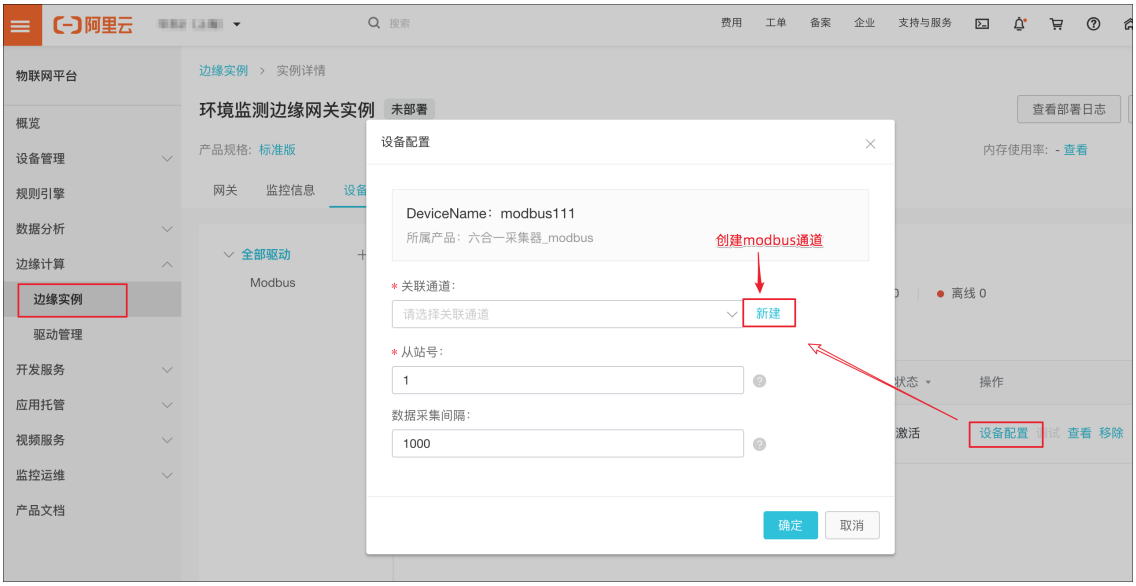
2. 安装Modbus设备驱动。边缘网关设备启动后，把Modbus数据解析为物模型JSON的驱动程序安装到Modbus设备。

i. 进入边缘计算**实例详情**页面的**设备驱动配置**页签，单击“+”图标，选择对应版本的官方Modbus驱动。如下图。



ii. 选中已添加的Modbus驱动，单击**分配子设备**，关联Modbus驱动和Modbus设备。

iii. 完成驱动和设备的关联后，单击设备名称右侧的**设备配置**，配置关联通道。参考下图操作。



iv. 在**编辑通道**页面，根据设备说明书填写通道配置参数，如下图。

v. 通道建立后，还需要配置传感器地址码（从站号），以及数据采集频率。

vi. 在**实例详情**页面右上角单击**部署**。这样就把设备驱动和配置参数从云端下发到边缘网关的设备上了。边缘计算网关就会按照已设定的频率从指定通道读取数据，并按照属性模型把Modbus数据转化成物模型JSON数据，上报到物联网平台。参考如下。

当部署状态显示为部署成功，表示部署实例完成。您可以单击**查看日志**，查看部署详情。

vii. 设备运行后，进入Modbus设备的**设备详情**页面，查看运行状态，可以看到传感器采集到的环境数据。

名词解释

- **驱动**

  Link IoT Edge中的设备接入模块称为驱动（Driver）或设备接入驱动。所有连接到Link IoT Edge的设备都需要通过驱动实现接入。

- **边缘实例**

  边缘实例通过网关关联您的设备，将设备接入到物联网平台进行管理控制，边缘实例同时也管理您设备使用的其他资源，例如驱动，函数计算，场景联动规则等。

- **部署**

  在云端为设备编排各种功能和配置，然后通过物联网边缘计算部署功能，将云端的能力和配置下发到边缘设备，实现设备功能和配置的自动运行。

- **监控信息**

  对边缘设备的状态、边缘能力运行状态进行监控，通过图表的方式展示在云端，随时随地监控边缘状态。

- **日志服务**

边缘设备上运行的各种能力均可以通过边缘API写入日志，同时支持该日志自动同步到云端，方便您随时随地查看日志状态。

- **消息路由**

  对边缘设备上数据流转做可视化的管理。通过路由规则动态规划消息的传输路径，使消息按照过滤条件，从消息源路由到目标节点的功能，称为消息路由。

- **产品规格**

  根据边缘场景不同，划分了三种不同的能力集合，称之为不同的产品规格，包括轻量版、标准版和专业版。

更多最佳实践

点击查看更多阿里云最佳实践。

9.Linux设备接入IoT平台

本文以在Ubuntu x86_64系统上编译设备端C语言SDK为例，介绍设备上云的配置和开发过程。

前提条件

在进行本示例配置前，您需要完成以下准备工作：

- 注册阿里云账号，并完成实名认证。
- 开通物联网平台。关于物联网平台介绍，参见物联网平台产品详情页。
- Ubuntu x86_64开发环境（PC或服务器）。
- 下载设备端C SDK Demo。

背景信息

阿里云物联网平台官方发布的设备端C语言版本SDK可以直接运行于Linux系统，并通过MQTT协议接入物联网平台。

创建产品和设备

在物联网平台注册产品和设备后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需配置到设备端SDK中。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

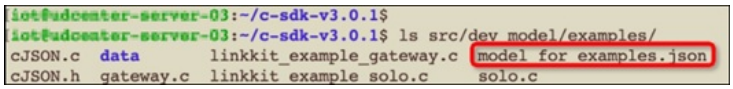
1. 登录物联网平台控制台。
2. 创建产品。
   i. 在左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，单击**创建产品**。
   iii. 填入产品信息，单击**确定**。完成产品创建。

3. 创建设备。
   i. 在左侧导航栏，选择**设备**。
   ii. 在**设备管理**页，单击**添加设备**。
   iii. 选择刚创建的产品，输入设备名称和备注名称，单击**确定**。完成设备创建。设备创建成功后，会弹出设备证书信息。您也可以在**设备管理**页，单击设备对应的**查看**按钮，进入**设备详情**页查看设备证书信息。

定义物模型

物模型指将物理空间中的实体进行数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能（包括属性、事件、服务）。完成功能定义后，系统将自动生成该产品的物模型。

1. 编辑物模型。物联网平台提供的设备端C SDK Demo包中，包含一个完整的物模型JSON文件。本示例中，将该JSON文件导入为已创建产品的物模型。
   i. 下载C SDK Demo包中的物模型JSON文件。
   ii. 解压Demo包后，打开*src/dev_model/examples*目录下的*model_for_examples.json*文件。



   iii. 将物模型JSON文件中的productKey的值替换为您在物联网平台上创建的产品ProductKey。

> **说明** C SDK Demo包中的物模型JSON文件中，productKey对应的值只是一个示例代码，您需将其替换为您的产品ProductKey。

   iv. 保存文件。
2. 导入物模型。将已经编辑好的物模型文件导入为产品的物模型。
   i. 在物联网平台控制台的左侧导航栏，选择**设备管理>产品**。
   ii. 在**产品管理**页，找到之前创建的产品，单击对应的**查看**按钮。
   iii. 在**产品详情**页**功能定义**页签下，单击**导入物模型**。
   iv. 在弹出的对话框中，单击 **上传文件**，上传上一步中编辑好的物模型JSON文件，单击**确定**。

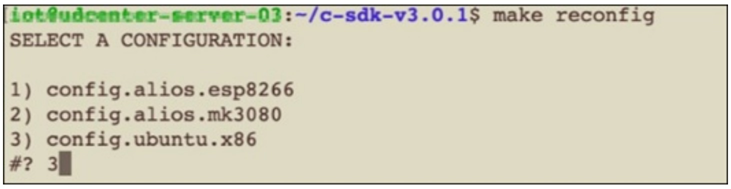
   导入成功后，该文件定义的所有功能将显示在自定义功能列表中。

配置C版本SDK

在开发工具中，导入Demo，并修改配置文件中的信息为您的设备信息。配置成功后，设备接入物联网平台。

1. 配置设备证书信息。在SDK Demo中*wrappers/os/ubuntu*目录下的*HAL_OS_linux.c*文件中，修改设备证书信息为您的设备证书信息。

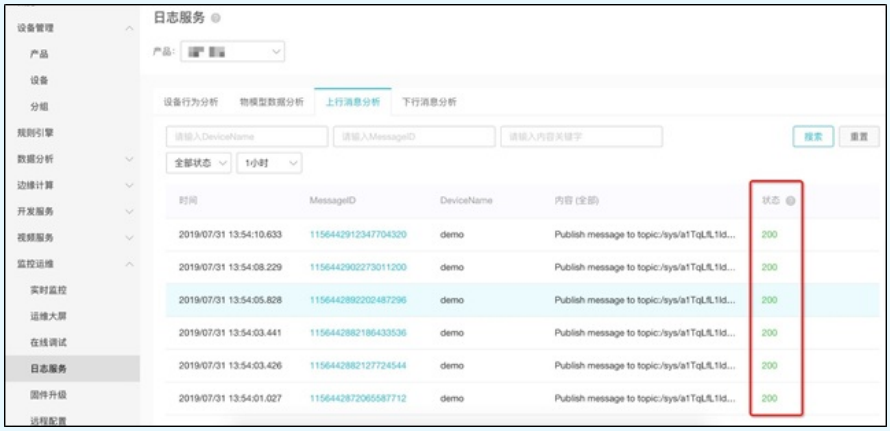
2. 编译SDK。在SDK根目录中，执行 `make reconfig` ，并选择3，然后make。



3. 运行测试。在SDK根目录中，执行 `./output/release/bin/linkkit-example-solo` 。执行结果如下图。

SDK运行成功后，可在物联网平台控制台上设备对应的设备详情页，查看设备状态和设备上报的物模型数据。

说明 您可以在物联网平台控制台上**监控运维>日志服务**页，搜索和查看设备日志。



名词解释

- **设备端SDK**

  阿里云物联网平台提供的Link Kit SDK即设备端SDK，用于设备端开发。设备需要支持TCP/IP协议栈才能集成Link Kit SDK。设备厂商将设备端SDK集成到设备上，设备便可通过该SDK安全地接入到阿里云物联网平台。目前，物联网平台提供六种设备端SDK：C SDK、Java SDK、Python SDK、Node.js SDK、Android SDK和iOS SDK。本示例使用的是C SDK。

  设备端SDK下载和使用说明，参见设备接入Link Kit SDK。

- **设备证书**

  设备证书指ProductKey、DeviceName和DeviceSecret，是阿里云物联网平台认证设备的标识。设备证书信息不可泄露。

- **ProductKey**

  物联网平台为产品颁发的全局唯一标识。

- **DeviceName**

  在注册设备时，自定义的或系统自动生成的设备名称，具备产品维度内的唯一性。

- **DeviceSecret**

  物联网平台为设备颁发的设备密钥。

更多最佳实践

点击查看更多阿里云最佳实践。