阿里云
物联网边缘计算

边缘开发指南

文档版本：20181107

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
	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。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 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。	`swich` *`{stand | slave}`*

目录

1 设备接入SDK

1.1 Nodejs版本SDK

设备接入SDK用于您在边缘计算节点上开发驱动，将设备连接到边缘计算节点，进而连接到物联网平台。设备接入SDK分为Node.js版本和Python版本，Node.js版本的SDK源码已开源，详情请见开源的*Node.js*库。

Node.js版本设备接入SDK使用说明

使用设备接入SDK前需要执行**`npm install linkedge-thing-access-sdk`**命令，安装SDK。

设备接入SDK的使用格式如下：

```
const {
    ThingAccessClient
} = require('linkedge-thing-access-sdk');
```

ThingAccessClient

设备接入客户端，您可以通过该客户端来主动上报设备属性或事件，也可被动接受云端下发的指令。

常量定义

名称	类型	描述
PRODUCT_KEY	String	配置对象（传给ThingAccessClient构造函数）的键值，指定云端分配的productKey。
DEVICE_NAME	String	配置对象（传给ThingAccessClient构造函数）的键值，指定云端分配的deviceName。
LOCAL_NAME	String	配置对象（传给ThingAccessClient构造函数）的键值，指定本地自定义的设备名。
RESULT_SUCCESS	Number	操作成功。用于回调函数返回状态码。
RESULT_FAILURE	Number	操作失败。用于回调函数返回状态码。
CALL_SERVICE	String	回调对象（传给ThingAccessClient构造函数）的键值，指定调用设备服务回调函数。
GET_PROPERTIES	String	回调对象（传给ThingAccessClient构造函数）的键值，指定获取设备属性回调函数。

名称	类型	描述
SET_PROPERTIES	String	回调对象（传给ThingAccessClient构造函数）的键值，指定设置设备属性回调函数。

ThingAccessClient(config, callbacks)

功能介绍

构造函数，使用指定的config和callbacks构造。

请求参数

名称	类型	描述
config	Object	元数据，用于配置该客户端。取值格式为： `{ "productKey": "Your Product Key", "deviceName": "Your Device Name" }`
callbacks	Object	回调函数对象。 • 指定设置设备属性的回调参数，请参见*callbacks.setProperties* • 指定获取设备属性的回调参数，请参见*callbacks.getProperties* • 指定调用设备服务的回调参数，请参见*callbacks.callService*

回调说明

```
callbacks: {
  setProperties: function(properties) {},
  getProperties: function(keys) {},
  callService: function(name, args) {}
}
```

callbacks.setProperties

功能介绍

设置具体设备属性的回调函数。通过回调函数，实现设置设备的属性。

请求参数

名称	类型	描述
properties	Object	设置属性的对象，取值格式为： `{ "key1": "value1", "key2": "value2" }`

callbacks.getProperties

功能介绍

获取具体设备属性的回调函数。通过回调函数，实现获取设备的属性。

请求参数

名称	类型	描述
keys	String[]	获取属性对应的名称，取值格式为： ['key1', 'key2']

callbacks.callService

功能介绍

调用设备服务回调函数。通过回调函数，实现调动设备服务。

请求参数

名称	类型	描述
name	String	设备服务名称。
args	Object	服务入参列表，取值格式为： `{ "key1": "value1", "key2": "value2" }`

setup() -> {Promise<Void>}

功能介绍

设备接入客户端初始化。该接口必须调用。

返回参数

```
Promise<Void>
```

registerAndOnline() -> {Promise<Void>}

功能介绍

将设备注册到边缘节点中并通知边缘节点上线设备。设备需要注册并上线后，设备端才能收到云端下发的指令或者发送数据到云端。

返回参数

```
Promise<Void>
```

reportEvent(eventName, args)

功能介绍

主动上报设备事件。

请求参数

名称	类型	描述
eventName	String	事件对应的名称，与您在产品定义中创建事件的名称一致。
args	Object	事件中包含的属性key与value，取值格式为： `{ "key1": "value1", "key2": "value2" }`

reportProperties(properties)

功能介绍

主动上报设备属性。

请求参数

名称	类型	描述
properties	Object	属性中包含的属性key与value，取值格式为： `{ "key1": "value1", "key2": "value2"`

名称	类型	描述
		}

cleanup() -> {Promise<Void>}

功能介绍

资源回收接口，您可以使用该接口回收您的资源。

返回参数

```
Promise<Void>
```

getTsl() -> {Promise<Void>}

功能介绍

返回TSL字符串，数据格式与云端一致。

返回参数

```
Promise<Void>
```

unregister() -> {Promise<Void>}

功能介绍

从边缘计算节点移除设备。请谨慎使用该接口。

返回参数

```
Promise<Void>
```

online() -> {Promise<Void>}

功能介绍

通知边缘节点设备上线，该接口一般在设备离线后再次上线时使用。

返回参数

```
Promise<Void>
```

offline() -> {Promise<Void>}

功能介绍

通知边缘节点设备已离线。

返回参数

```
Promise<Void>
```

1.2 Python版本SDK

本章为您介绍Python版本的SDK使用方法及相关API。Link IoT Edge提供Python版本的SDK，名称为*lethingaccesssdk*。

Python版本设备接入SDK使用说明

在使用设备接入SDK前需要在您的代码库中导入**import lethingaccesssdk**。

如下示例展示了使用Python版本的SDK进行开发操作：

```
import lethingaccesssdk
import time

class Demo_device(lethingaccesssdk.ThingCallback):
  def __init__(self):
    self.LightSwitch = 1

  def callService(self, name, input_value):
    pass

  def getProperties(self, input_value):
    pass

  def setProperties(self, input_value):
    pass

app_callback = Demo_device()
client = lethingaccesssdk.ThingAccessClient("YourProductKey", "
YourDeviceName")
client.registerAndonline(app_callback)

while True:
  time.sleep(3)
  propertiesDict={"LightSwitch":app_callback.LightSwitch}
  client.reportProperties(propertiesDict)

def handler(event, context):

  return 'hello world'
```

ThingCallback

需要先实现一个类来继承ThingCallback，然后再实现*setProperties*、*getProperties*和*callService*三个函数。

setProperties

功能介绍

设置具体设备属性函数。

请求参数

名称	类型	描述
properties	Object	设置属性的对象，取值格式为： `{ "key1": "value1", "key2": "value2" }`

getProperties

功能介绍

获取具体设备属性的函数。

请求参数

名称	类型	描述
keys	String[]	获取属性对应的名称，取值格式为： ['key1', 'key2']

callService

功能介绍

调用设备服务函数。

请求参数

名称	类型	描述
name	String	设备服务名称。
args	Object	服务入参列表，取值格式为： `{ "key1": "value1", "key2": "value2" }`

ThingAccessClient

设备接入客户端，您可以通过该客户端来主动上报设备属性或事件，也可被动接受云端下发的指令。

ThingAccessClient(productKey，deviceName)

功能介绍

构造函数，使用您自己设备的指定的ProductKey和DeviceName构造。

registerAndOnline(ThingCallback object)

功能介绍

将设备注册到边缘节点中并通知边缘节点上线设备。设备需要注册并上线后，设备端才能收到云端下发的指令或者发送数据到云端。

reportEvent(eventName, args)

功能介绍

主动上报设备事件。

请求参数

名称	类型	描述
eventName	String	事件对应的名称，与您在产品定义中创建事件的名称一致。
args	Object	事件中包含的属性key与value，取值格式为： `{ "key1": "value1", "key2": "value2" }`

reportProperties(properties)

功能介绍

主动上报设备属性。

请求参数

名称	类型	描述
properties	Object	属性中包含的属性key与value，取值格式为： `{ "key1": "value1", "key2": "value2"`

名称	类型	描述
		}

getTsl()

功能介绍

返回TSL字符串，数据格式与云端一致。

返回参数

```
TSL字符串
```

unregister()

功能介绍

从边缘计算节点移除设备。请谨慎使用该接口。

online()

功能介绍

通知边缘节点设备上线，该接口一般在设备离线后再次上线时使用。

offline()

功能介绍

通知边缘节点设备已离线。

2 设备接入SDK综合示例

本文展示一段使用设备接入SDK，开发驱动的代码片段。设备使用thing标识，拥有一个temperature的只读属性，在连接到边缘计算节点后，每隔2秒向平台上报属性和高温事件。

Node.js版本

```
/*
 * Copyright (c) 2018 Alibaba Group Holding Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

'use strict';

const {
  RESULT_SUCCESS,
  RESULT_FAILURE,
  ThingAccessClient,
} = require('linkedge-thing-access-sdk');

var driverConfig;
try {
  driverConfig = JSON.parse(process.env.FC_DRIVER_CONFIG)
} catch (err) {
  throw new Error('The driver config is not in JSON format!');
}
var configs = driverConfig['deviceList'];
if (!Array.isArray(configs) || configs.length === 0) {
  throw new Error('No device is bound with the driver!');
}

var args = configs.map((config) => {
  var self = {
    config,
    thing: {
      temperature: 41,
    },
    callbacks: {
      setProperties: function (properties) {
        // Usually, in this callback we should set properties to the
physical thing and
        // return the result. Here we just return a failed result
since the properties
        // are read-only.
        console.log('Set properties %s to thing %s-%s', JSON.stringify
(properties),
```

```
          config.productKey, config.deviceName);
        // Return an object representing the result in the following
form or the promise
        // wrapper of the object.
        return {
          code: RESULT_FAILURE,
          message: 'failure',
        };
      },
      getProperties: function (keys) {
        // Usually, in this callback we should get properties from the
 physical thing and
        // return the result. Here we return the simulated properties.
        console.log('Get properties %s from thing %s-%s', JSON.
stringify(keys),
          config.productKey, config.deviceName);
        // Return an object representing the result in the following
form or the promise
        // wrapper of the object.
        if (keys.includes('temperature')) {
          return {
            code: RESULT_SUCCESS,
            message: 'success',
            params: {
              temperature: self.thing.temperature,
            }
          };
        }
        return {
          code: RESULT_FAILURE,
          message: 'The requested properties does not exist.',
        }
      },
      callService: function (name, args) {
        // Usually, in this callback we should call services on the
physical thing and
        // return the result. Here we just return a failed result
since no service
        // provided by the thing.
        console.log('Call service %s with %s on thing %s-%s', JSON.
stringify(name),
          JSON.stringify(args), config.productKey, config.deviceName);
        // Return an object representing the result in the following
form or the promise
        // wrapper of the object
        return new Promise((resolve) => {
          resolve({
            code: RESULT_FAILURE,
            message: 'The requested service does not exist.',
          })
        });
      }
    },
  };
  return self;
});

args.forEach((item) => {
  var client = new ThingAccessClient(item.config, item.callbacks);
  client.setup()
    .then(() => {
      return client.registerAndOnline();
```

```
    })
    .then(() => {
      // Push events and properties to LinkEdge platform.
      return new Promise(() => {
        setInterval(() => {
          var temperature = item.thing.temperature;
          if (temperature > 40) {
            client.reportEvent('high_temperature', {'temperature':
temperature});
          }
          client.reportProperties({'temperature': temperature});
        }, 2000);
      });
    })
    .catch(err => {
      console.log(err);
      return client.cleanup();
    })
    .catch(err => {
      console.log(err);
    });
});

module.exports.handler = function (event, context, callback) {
  console.log(event);
  console.log(context);
  callback(null);
};
```

Python版本

```
# -*- coding: utf-8 -*-
import lethingaccesssdk
import time
import json
import os
import logging


# Base on device, user need to implement the callService, getPropert
ies and getProperties function.
class Temperature_device(lethingaccesssdk.ThingCallback):
  def __init__(self):
    self.temperature = 41

  def callService(self, name, input_value):
    return -1, {}

  def getProperties(self, input_value):
    if input_value[0] == "temperature":
      return 0, {input_value[0]: self.LightSwitch}
    else:
      return -1, {}

  def setProperties(self, input_value):
    if "temperature" in input_value:
      self.LightSwitch = input_value["temperature"]
      return 0, {}

device_obj_dict = {}
try:
```

```
  driver_conf = json.loads(os.environ.get("FC_DRIVER_CONFIG"))
  if "deviceList" in driver_conf:
    device_list_conf = driver_conf["deviceList"]
    device = device_list_conf[0]
    pk = device["productKey"]
    dn = device["deviceName"]
    app_callback = Temperature_device()
    client = lethingaccesssdk.ThingAccessClient(pk, dn)
    client.registerAndonline(app_callback)
    while True:
      time.sleep(2)
      if app_callback.temperature > 40:
        client.reportEvent('high_temperature', {'temperature':
app_callback.temperature})
        client.reportProperties({'temperature': app_callback.
temperature})
except Exception as e:
  logging.error(e)

# don't remove this function
def handler(event, context):
  return 'hello world'
```

3 Link IoT Edge核心SDK

3.1 Nodejs版本SDK

3.1.1 IoTData

Link IoT Edge核心SDK允许开发人员使用Node.js编写FC函数。

包含IoT操作相关API的类。

publish(params, callback)

发布指定消息。您可以通过消息路由功能进一步设置消息的流转路径。

名称	类型	描述
params	Object	参数对象，包含如下必选参数。 • topic：{String}，消息主题。 • payload：{String}，消息负载。
callback(err)	function	回调函数。遵循JavaScript标准实践。成功则err为null，否则err包含发生的错误。

getThingProperties(params, callback)

获取指定的设备属性。

名称	类型	描述
params	Object	参数对象，包含如下必选参数。 • productKey：{String}，设备的ProductKey，创建设备时生成。 • deviceName：{String}，设备的DeviceName，创建设备时生成。 • payload：{Array}，包含设备属性的数组。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。 • err：{Error}，成功则为null，否则为发生的错误。 • data：{Object}，指定设备属性的键值。

setThingProperties(params, callback)

设置指定的设备属性。

名称	类型	描述
params	Object	参数对象，包含如下必选参数。 • productKey：{String}，设备的ProductKey，创建设备时生成。 • deviceName：{String}，设备的DeviceName，创建设备时生成。 • payload：{Object}，包含设置给设备的属性键值。
callback(err)	function	回调函数。遵循JavaScript标准实践。 • err：{Error}，成功则返回null，否则为发生的错误。

callThingService(params, callback)

调用指定的设备服务。

名称	类型	描述
params	Object	参数对象，包含如下参数。 • productKey：{String} (Required)，设备的ProductKey，创建设备时生成。 • deviceName：{String} (Required)，设备的DeviceName，创建设备时生成。 • service：{String} (Required)，被调用的服务名。 • payload：{String|Buffer}，传给服务的参数。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。 • err：{Error}，成功则返回null，否则为发生的错误。 • data：被调用服务的返回值。

getThingsWithTags(params, callback)

获取满足所有标签的设备信息。

名称	类型	描述
params	Object	参数对象，包含如下必选参数。 • payload：{Array}，一个{<标签名>:<标签值>}组成的数组。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。 • err：{Error}，成功则返回null，否则为发生的错误。 • data：{Array}，满足条件的设备信息数组。

3.1.2 FCClient

包含函数计算相关API的类。

invokeFunction(params, callback)

调用指定函数。

名称	类型	描述
params	Object	参数对象，包含如下参数。 • functionId：{String}，被调用函数的唯一ID。您还可以通过serviceName与functionName同时指定被调函数，无需使用functionId。 • serviceName：{String}，服务名，必须与functionName共同指定被调函数。您也可以通过functionId指定被调函数，无需使用serviceName。 • functionName：{String}，函数名，必须与serviceName共同指定被调函数。您也可以通过functionId指定被调函数，无需使用functionName。 • invocationType：{String} 调用类型。同步为Sync，异步为Async。默认为同步。 • payload：{String|Buffer} 参数信息作为函数的输入。
callback(err, data)	function	回调函数。遵循JS标准实践。 • err：{Error} 成功则返回null，否则为发生的错误。

名称	类型	描述
		• data：被调函数的返回值。

3.2 Python版本SDK

3.2.1 IoTData

Link IoT Edge核心SDK同时允许开发人员使用Python编写FC函数。

包含IoT操作相关API的类。

publish(params)

发布指定消息。您可以通过消息路由功能进一步设置消息的流转路径。

名称	类型	描述
params	Dict	参数对象，包含如下必选参数。 • topic：{String}，消息主题。 • payload：{String}，消息负载。

getThingProperties(params)

获取指定的设备属性。

名称	类型	描述
params	Dict	参数对象，包含如下必选参数。 • productKey：{String}，设备的ProductKey，创建设备时生成。 • deviceName：{String}，设备的DeviceName，创建设备时生成。 • payload：{List}，包含设备属性的数组。
return	Dict	指定设备属性的键值。

setThingProperties(params)

设置指定的设备属性。

名称	类型	描述
params	Dict	参数对象，包含如下必选参数。

名称	类型	描述
		• productKey：{String}，设备的ProductKey，创建设备时生成。 • deviceName：{String}，设备的DeviceName，创建设备时生成。 • payload：{Dict}，包含设置给设备的属性键值。
return	Dict	成功则返回True，否则为发生的错误。

callThingService(params)

调用指定的设备服务。

名称	类型	描述
params	Dict	参数对象，包含如下参数。 • productKey：{String} (Required)，设备的ProductKey，创建设备时生成。 • deviceName：{String} (Required)，设备的DeviceName，创建设备时生成。 • service：{String} (Required)，被调用的服务名。 • payload：{String|Bytes}，传给服务的参数。
return	Dict	被调用服务的返回值。

getThingsWithTags(params)

获取满足所有标签的设备信息。

名称	类型	描述
params	Dict	参数对象，包含如下必选参数。 • payload：{List}，一个由{<标签名>:<标签值>}组成的数组。
return	List	满足条件的设备信息数组。

3.2.2 Client

包含函数计算相关API的类。

invoke_function(params)

调用指定函数。

名称	类型	描述
params	Dict	参数对象，包含如下参数。 • functionId：{String}，被调用函数的唯一ID。您还可以通过serviceName与functionName同时指定被调函数，无需使用functionId。 • serviceName：{String}，服务名，必须与functionName共同指定被调函数。您也可以通过functionId指定被调函数，无需使用serviceName。 • functionName：{String}，函数名，必须与serviceName共同指定被调函数。您也可以通过functionId指定被调函数，无需使用functionName。 • invocationType：{String}，调用类型。同步为Sync，异步为Async。默认为同步。 • payload：{String|Bytes}，参数信息作为函数的输入。
return	Dict	被调函数的返回值。

4 Link IoT Edge核心SDK综合示例

本文展示使用Link IoT Edge核心SDK控制灯设备的操作代码。示例中，当光照传感器上报的光照度超过500时关闭灯，光照度不足100时打开灯。

Node.js版本综合示例

```
/*
 * Copyright (c) 2018 Alibaba Group Holding Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * The example demonstrates closing a light when it monitor the
illuminance
 * reported by a light sensor is greater than 500.
 */

'use strict';

const leSdk = require('linkedge-core-sdk');

const iotData = new leSdk.IoTData();

const productKey = 'Light Product Key';
const deviceName = 'Light Device Name';

module.exports.handler = function (event, context, callback) {

  console.log(`LightMonitor is invoking with ${event.toString()}.`);

  var illuminance;
  try {
    var obj = JSON.parse(event.toString());
    if (!obj.payload) {
      callback(new Error(`Can't find "payload" in event.`));
      return;
    }
    var payload = JSON.parse(obj.payload);
    if (!payload['MeasuredIlluminance']) {
      callback(new Error(`Can't find "MeasuredIlluminance" in event
.`));
      return;
    }
    illuminance = payload['MeasuredIlluminance'].value || 0;
  } catch (err) {
```

```
    err = new Error(`Parse event failed due to ${err}.`);
    callback(err);
    return;
  }

  if (illuminance > 500) {
    turnOff(callback);
  } else if (illuminance <= 100) {
    turnOn(callback);
  } else {
    console.log(`Illuminance value is ${illuminance}, ignore.`);
    callback(null);
  }
};

function turnOff(callback) {
  // Turn off the light according to product key and device name.
  iotData.setThingProperties({
    productKey,
    deviceName,
    payload: {'LightSwitch': 0},
  }, function (err) {
    if (err) {
      console.log(`Failed to turn off the light due to ${err}.`);
      callback(err);
    } else {
      console.log(`Turns off light successfully.`);
      callback(null);
    }
  });
}

function turnOn(callback) {
  // Turn on the light according to product key and device name.
  iotData.setThingProperties({
    productKey,
    deviceName,
    payload: {'LightSwitch': 1},
  }, function (err) {
    if (err) {
      console.log(`Failed to turn on the light due to ${err}.`);
      callback(err);
    } else {
      console.log(`Turns on light successfully.`);
      callback(null);
    }
  });
}
```

Python版本综合示例

```
# -*- coding: utf-8 -*-
import lecoresdk
import json


ON = 1
OFF = 0

def light_turn(status):
  it = lecoresdk.IoTData()
```

```
  set_params = {"productKey": "Light Product Key",
                            "deviceName": "Light Device Name",
                            "payload": {"LightSwitch":status}}
  res = it.setThingProperties(set_params)

def handler(event, context):
  event_json = json.loads(event.decode("utf-8"))
  if "payload" in event_json:
    payload_json = json.loads(event_json["payload"])
    if "illuminance" in payload_json and "value" in payload_json["
illuminance"]:
       illuminance = payload_json["illuminance"]["value"]
       if illuminance > 500:
          light_turn(OFF)
       elif illuminance <= 100:
          light_turn(ON)
  return 'hello world'
```

5 云端开发指南

5.1 设备接入开发