ALIBABA CLOUD

阿里云

物联网边缘计算
边缘端开发指南

文档版本：20220119

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.设备接入SDK

1.1. C版本SDK

本章为您介绍C版本的SDK使用方法及相关API。Link IoT Edge提供C版本的SDK，名称为*linkedge-thing-access-sdk-c*。

C版本开源的SDK源码请参见开源C库。

get_properties_callback

```
/*
     * 获取属性（对应设备产品物模型属性定义）的回调函数，需驱动开发者实现获取属性业务逻辑。
     *
     * Link IoT Edge需要获取某个设备的属性时，SDK会调用该接口间接获取到数据并封装成固定格式后回传给Link IoT Edge。
     * 开发者需要根据设备id和属性名找到设备，将获取到的属性值按照@device_data_t格式填充。
     *
     * @dev_handle:          Link IoT Edge需要获取属性的具体某个设备。
     * @properties:          属性值键值结构，驱动开发者需要将根据属性名称获取到的属性值更新到properties中。
     * @properties_count:    属性个数。
     * @usr_data:            注册设备时，用户传递的私有数据。
     * 所有属性均获取成功则返回LE_SUCCESS，其他则返回错误码（参考le_error.h错误码宏定义）。
     * */
typedef int (*get_properties_callback)(device_handle_t dev_handle,
                                       leda_device_data_t properties[],
                                       int properties_count,
                                       void *usr_data);
```

set_properties_callback

```
/*
     * 设置属性（对应设备产品物模型属性定义）的回调函数，需驱动开发者实现设置属性业务逻辑。
     *
     * Link IoT Edge需要设置某个设备的属性时，SDK会调用该接口将具体的属性值传递给应用程序，开发者需要在本回调
     * 函数里将属性设置到设备。
     *
     * @dev_handle:          Link IoT Edge需要设置属性的具体某个设备。
     * @properties:          Link IoT Edge需要设置的设备的属性名称和值。
     * @properties_count:    属性个数。
     * @usr_data:            注册设备时，用户传递的私有数据。
     *
     * 若获取成功则返回LE_SUCCESS，失败则返回错误码（参考le_error.h错误码宏定义）。
     * */
typedef int (*set_properties_callback)(device_handle_t dev_handle,
                                       const leda_device_data_t properties[],
                                       int properties_count,
                                       void *usr_data);
```

call_service_callback

```
/*
     * 服务（对应设备产品物模型服务定义）调用的回调函数，需要驱动开发者实现服务对应业务逻辑。
     *
     * Link IoT Edge需要调用某个设备的服务时，SDK会调用该接口将具体的服务参数传递给应用程序，开发者需要在本回调
     * 函数里调用具体的服务，并将服务返回值按照@device_data_t格式填充到output_data。
     *
     * @dev_handle:    Link IoT Edge需要调用服务的具体某个设备。
     * @service_name:  Link IoT Edge需要调用的设备的具体某个服务名，名称与设备产品物模型一致。
     * @data:          Link IoT Edge需要调用的设备的具体某个服务参数，参数与设备产品物模型保持一致。
     * @data_count:    Link IoT Edge需要调用的设备的具体某个服务参数个数。
     * @output_data:   开发者需要将服务调用的返回值，按照设备产品物模型规定的服务格式返回到output中。
     * @usr_data:      注册设备时，用户传递的私有数据。
     *
     * 若获取成功则返回LE_SUCCESS，失败则返回错误码（参考le_error.h错误码宏定义）。
     * */
typedef int (*call_service_callback)(device_handle_t dev_handle,
                                     const char *service_name,
                                     const leda_device_data_t data[],
                                     int data_count,
                                     leda_device_data_t output_data[],
                                     void *usr_data);
```

leda_report_properties

```
/*
 * 上报属性，在设备所属产品物模型中规定了设备的属性上报能力。
 *
 * 上报属性，可以上报一个，也可以多个一起上报。
 *
 * dev_handle:          设备在Link IoT Edge本地唯一标识。
 * properties:          @leda_device_data_t，属性数组。
 * properties_count:    本次上报属性个数。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 */
int leda_report_properties(device_handle_t dev_handle, const leda_device_data_t properties[], int properties_count);
```

leda_report_event

```
/*
 * 上报事件，设备具有的事件上报能力在设备产品物模型有规定。
 *
 *
 * dev_handle:   设备在Link IoT Edge本地唯一标识。
 * event_name:   事件名称。
 * data:         @leda_device_data_t，事件参数数组。
 * data_count:   事件参数数组长度。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 */
int leda_report_event(device_handle_t dev_handle, const char *event_name, const leda_device
_data_t data[], int data_count);
```

leda_offline

```
/*
 * 下线设备，假如设备工作在不正常的状态或设备退出前，可以先下线设备，这样Link IoT Edge就不会继续下发消
息到设备侧。
 *
 * dev_handle:   设备在Link IoT Edge本地唯一标识。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 */
int leda_offline(device_handle_t dev_handle);
```

leda_online

```
/*
 * 上线设备，设备只有上线后，才能被Link IoT Edge识别。
 *
 * dev_handle:   设备在Link IoT Edge本地唯一标识。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
int leda_online(device_handle_t dev_handle);
```

leda_register_and_online_by_device_name

```
/*
 * 通过已在阿里云物联网平台创建的设备device_name，注册并上线设备，申请设备唯一标识符。
 *
 * 若需要注册多个设备，则多次调用该接口即可。
 *
 * product_key:  在阿里云物联网平台创建的产品ProductKey。
 * device_name:  在阿里云物联网平台创建的设备名称DeviceName。
 * device_cb:    请求调用设备回调函数结构体，详细描述见@leda_device_callback。
 * usr_data:     设备注册时传入私有数据，在回调中会传给设备。
 *
 * 阻塞接口，返回值设备在Link IoT Edge本地唯一标识，>= 0表示有效，< 0 表示无效。
 *
 */
device_handle_t leda_register_and_online_by_device_name(const char *product_key, const char
*device_name, leda_device_callback_t *device_cb, void *usr_data);
```

leda_register_and_online_by_local_name

```
/*
 * 通过本地自定义设备名称，注册并上线设备，申请设备唯一标识符。
 *
 * 若需要注册多个设备，则多次调用该接口即可。
 *
 * product_key:  在阿里云物联网平台创建的产品ProductKey。
 * local_name:   由设备特征值组成的唯一描述信息，必须保证同一个product_key时，每个设备名称不同。
 * device_cb:    请求调用设备回调函数结构体，详细描述见@leda_device_callback。
 * usr_data:     设备注册时传入私有数据，在回调中会传给设备。
 *
 * 阻塞接口，返回值设备在Link IoT Edge本地唯一标识，>= 0表示有效，< 0 表示无效。
 *
 * 注：在同一产品ProductKey条件设备注册，不允许本接口和leda_register_and_online_by_device_name接
口同时使用。
 * 即每一个产品ProductKey设备注册必须使用同一接口，否则设备注册会发生不可控行为。
 */
device_handle_t leda_register_and_online_by_local_name(const char *product_key, const char
*local_name, leda_device_callback_t *device_cb, void *usr_data);
```

leda_init

```
/*
 * 驱动模块初始化，模块内部会创建工作线程池，异步执行阿里云物联网平台下发的设备操作请求，工作线程数目通
过worker_thread_nums配置。
 *
 * worker_thread_nums:  线程池工作线程数，该数值根据注册设备数量进行设置。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
int leda_init(int worker_thread_nums);
```

leda_exit

```
/*
 * 驱动模块退出。
 *
 * 模块退出前，释放资源。
 *
 * 阻塞接口。
 */
void leda_exit(void);
```

leda_get_driver_info_size

```
/*
 * 获取驱动信息长度。
 *
 * 阻塞接口，成功返回驱动信息长度，失败返回0。
 */
int leda_get_driver_info_size(void);
```

leda_get_driver_info

```
/*
 * 获取驱动信息（在物联网平台配置的驱动配置）。
 *
 * driver_info:  驱动信息，需要提前申请好内存传入。
 * size:         驱动信息长度，leda_get_driver_info_size，如果传入driver_info比实际配置内容长度短，会返回LE_ERROR_INVAILD_PARAM。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 * 配置格式:
    {
        "json":{
            "ip":"127.0.0.1",
            "port":54321
        },
        "kv":[
            {
                "key":"ip",
                "value":"127.0.0.1",
                "note":"ip地址"
            },
            {
                "key":"port",
                "value":"54321",
                "note":"port端口"
            }
        ],
        "fileList":[
            {
                "path":"device_config.json"
            }
        ]
    }
 */
int leda_get_driver_info(char *driver_info, int size);
```

leda_get_device_info_size

```
/*
 * 获取设备信息长度。
 *
 * 阻塞接口，成功返回设备信息长度，失败返回0。
 */
int leda_get_device_info_size(void);
```

leda_get_device_info

```
/*
 * 获取设备信息（在物联网平台配置的设备配置）。
 *
 * device_info:    设备信息，需要提前申请好内存传入。
 * size:           设备信息长度，该长度通过leda_get_device_info_size接口获取，如果传入device_info
比实际配置内容长度短，会返回LE_ERROR_INVAILD_PARAM。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 * 配置格式:
    [
        {
            "custom":{
                "port":12345,
                "ip":"127.0.0.1"
            },
            "deviceName":"device1",
            "productKey":"a1ccxxeypky"
        }
    ]
 */
int leda_get_device_info(char *device_info, int size);
```

leda_get_config_size

```
/*
 * 获取驱动配置长度。
 *
 * 阻塞接口，成功返回驱动配置长度，失败返回0。
 */
int leda_get_config_size(void);
```

leda_get_config

```
/*
 * 获取驱动所有配置。
 *
 * config:          驱动配置，需要提前申请好内存传入。
 * size:            驱动配置长度，该长度通过leda_get_config_size接口获取，如果传入config比实际配置内
容长度短，会返回LE_ERROR_INVAILD_PARAM。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 *
 * 配置格式:
    {
        "config":{
            "json":{
                "ip":"127.0.0.1",
                "port":54321
            },
            "kv":[
                {
                    "key":"ip",
                    "value":"127.0.0.1",
                    "note":"ip地址"
                },
                {
                    "key":"port",
                    "value":"54321",
                    "note":"port端口"
                }
            ],
            "fileList":[
                {
                    "path":"device_config.json"
                }
            ]
        },
        "deviceList":[
            {
                "custom":"{"port":12345,"ip":"127.0.0.1"}",
                "deviceName":"device1",
                "productKey":"a1ccxxeypky"
            }
        ]
    }
 */
int leda_get_config(char *config, int size);
```

config_changed_callback

```
/*
 * 驱动配置变更回调接口。
 *
 * config:          配置信息。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
typedef int (*config_changed_callback)(const char *config);
```

leda_register_config_changed_callback

```
/*
 * 订阅驱动配置变更监听回调。
 *
 * config_cb:       配置变更通知回调接口。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
int leda_register_config_changed_callback(config_changed_callback config_cb);
```

leda_get_tsl_size

```
/*
 * 获取指定产品ProductKey对应物模型内容长度。
 *
 * product_key:     产品ProductKey。
 *
 * 阻塞接口，成功返回product_key对应物模型内容长度，失败返回0。
 */
int leda_get_tsl_size(const char *product_key);
```

leda_get_tsl

```
/*
 * 获取指定产品ProductKey对应物模型内容。
 *
 * product_key:     产品ProductKey。
 * tsl:             物模型内容，需要提前申请好内存传入。
 * size:            物模型内容长度，该长度通过leda_get_tsl_size接口获取，如果传入tsl比实际物模型内容
长度短，会返回LE_ERROR_INVAILD_PARAM。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
int leda_get_tsl(const char *product_key, char *tsl, int size);
```

leda_get_tsl_ext_info_size

```
/*
 * 获取指定产品ProductKey对应物模型扩展信息内容长度。
 *
 * product_key:    产品ProductKey。
 *
 * 阻塞接口，成功返回product_key对应物模型扩展信息内容长度，失败返回0。
 */
int leda_get_tsl_ext_info_size(const char *product_key);
```

leda_get_tsl_ext_info

```
/*
 * 获取指定产品ProductKey对应物模型扩展信息内容。
 *
 * product_key:    产品ProductKey。
 * tsl_ext_info:   物模型扩展信息，需要提前申请好内存传入。
 * size:           物模型扩展信息长度，该长度通过leda_get_tsl_ext_info_size接口获取，如果传入tsl_e
xt_info比实际物模型扩展信息内容长度短，会返回LE_ERROR_INVAILD_PARAM。
 *
 * 阻塞接口，成功返回LE_SUCCESS，失败返回错误码。
 */
int leda_get_tsl_ext_info(const char *product_key, char *tsl_ext_info, int size);
```

leda_get_device_handle

```
/*
 * 获取设备句柄。
 *
 * product_key:  产品ProductKey。
 * device_name:  设备名称DeviceName。
 *
 * 阻塞接口，成功返回device_handle_t，失败返回小于0数值。
 */
device_handle_t leda_get_device_handle(const char *product_key, const char *device_name);
```

1.2. Nodejs版本SDK

设备接入SDK用于您在网关上开发驱动，将设备连接到网关，进而连接到物联网平台。

Node.js版本开源的SDK源码请见开源的Node.js库。

安装和使用

1. 您可以通过如下命令来安装SDK。

   ```
   npm install linkedge-thing-access-sdk
   ```

2. 安装完成后，您可以根据SDK接口进行驱动开发。

   > 注意 完成驱动开发后，直接运行会提示错误，必须通过物联网平台控制台，将已开发的驱动部署到网关中方可执行。部署驱动到网关的操作请参考驱动开发。

使用SDK开发驱动的示例代码片段如下所示。

```
const {
  Config,
  ThingAccessClient
} = require('linkedge-thing-access-sdk');
const callbacks = {
  setProperties: function (properties) {
    // Set properties to the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return {
      code: 0,
      message: 'success',
    };
  },
  getProperties: function (keys) {
    // Get properties from the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return {
      code: 0,
      message: 'success',
      params: {
        key1: 'value1',
        key2: 'value2',
      }
    };
  },
  callService: function (name, args) {
    // Call services on the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return new Promise((resolve) => {
      resolve({
        code: 0,
        message: 'success',
      });
    });
  }
};
Config.get()
  .then(config => {
    const thingInfos = config.getThingInfos();
    thingInfos.forEach(thingInfo => {
      const client = new ThingAccessClient(thingInfo, callbacks);
      client.registerAndOnline()
        .then(() => {
          return new Promise(() => {
            setInterval(() => {
              client.reportEvent('high_temperature', { temperature: 41 });
              client.reportProperties({ 'temperature': 41 });
            }, 2000);
          });
        })
        .catch(err => {
          console.log(err);
```

```
            console.log(err);
            client.cleanup();
          });
          .catch(err => {
            console.log(err);
          });
      });
    });
```

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。
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。回调函数格式请参见callbacks.callService()。
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。回调函数格式请参见callbacks.getProperties()。
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。回调函数格式请参见callbacks.setProperties()。
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。
ERROR_CLEANUP	String	调用cleanup()出错错误码。
ERROR_CONNECT	String	调用registerAndOnline()出错错误码。
ERROR_DISCONNECT	String	调用offline()出错错误码。
ERROR_GET_CONFIG	String	调用getConfig()出错错误码。
ERROR_GET_TSL	String	调用getTsl()出错错误码。
ERROR_GET_TSL_EXT_INFO	String	调用getTslExtInfo()出错错误码。
ERROR_UNREGISTER	String	调用unregister()出错错误码。

Config

驱动相关配置信息。

- static get()

  返回全局的驱动配置对象，该配置通常在设备与驱动程序关联时由系统自动生成。

  > **说明** 该接口的Config.get()调用方法与getConfig()接口的区别在于，getConfig()返回配置字符串，Config.get()返回配置对象。

  返回值：

  ```
  Promise<Config>
  ```

- static registerChangedCallback(callback)

  注册配置变更回调函数。

  请求参数

名称	类型	描述
callback(String)	Function	回调函数，配置变更时被调用。

- static unregisterChangedCallback(callback)

  注销配置变更回调函数。

  请求参数

名称	类型	描述
callback(String)	Function	回调函数，配置变更时被调用。

- Config(string)

  基于配置字符串构造新的Config对象。

  请求参数

名称	类型	描述
string	String	JSON配置字符串。

- getThingInfos()

  返回所有设备相关信息。

  返回值：

  ```
  ThingInfo[]
  ```

- getDriverInfo()

  返回驱动相关信息。

  返回值：

  ```
  Object
  ```

ThingInfo

此类（结构）接口用于标识连接到Link IoT Edge的设备信息，即某个设备连接到了Link IoT Edge，那么可以通过此类接口标识该设备的productKey, deviceName, 自定义配置等信息。

ThingInfo(productKey, deviceName, custom)

构造一个新的ThingInfo对象。

请求参数

名称	类型	描述
productKey	String	产品唯一标识。
deviceName	String	设备名称。
custom	Object	设备自定义配置。

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(config, callbacks)

  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。取值格式为： `callbacks: { setProperties: function(properties) {}, getProperties: function(keys) {}, callService: function(name, args) {} }` ○ 指定设置设备属性的回调参数，请参见本文下方callbacks.setProperties内容 ○ 指定获取设备属性的回调参数，请参见本文下方callbacks.getProperties内容 ○ 指定调用设备服务的回调参数，请参见本文下方callbacks.callService内容

- callbacks.setProperties(properties)

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" }`

返回值：

```
{
  "code": 0,
  "message": "string",
  "params": {}
}
```

返回参数

名称	类型	描述
code	Number	状态码。 ○ 0：接口调用成功。 ○ 非0：接口调用失败，报非0值对应的错误。
message	String	可选参数，状态描述信息。
params	Object	可选参数，用于返回每个属性的设置结果，其值为每个属性设置后的实际值。

- callbacks.getProperties(keys)

  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']`

  返回值：

```
{
  "code": 0,
  "message": "string",
  "params": {}
}
```

返回参数

名称	类型	描述
code	Number	状态码。 ○ 0：接口调用成功。 ○ 非0：接口调用失败，报非0值对应的错误。
message	String	可选参数，状态描述信息。
params	Object	可选参数，用于获取属性成功时，返回对应的属性值。

- callbacks.callService(name, args)

  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" }`

  返回参数

名称	类型	描述
code	Number	状态码。 ○ 0：接口调用成功。 ○ 非0：接口调用失败，报非0值对应的错误。
message	String	可选参数，状态描述信息。
params	Object	可选参数，用于调用设备服务成功时，返回额外的信息。

- setup()

  设备接入客户端初始化。

  > 注意 Link IoT Edge当前版本已不再使用该接口，之前遗留的调用不受影响。

  返回值：

```
Promise<Void>
```

- registerAndOnline()

  将设备注册到网关中并通知网关上线设备。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。

  返回值：

  ```
  Promise<Void>
  ```

- online()

  通知网关设备已上线，该接口一般在设备离线后再次上线时使用。

  返回值：

  ```
  Promise<Void>
  ```

- offline()

  通知网关设备已离线。

  返回值：

  ```
  Promise<Void>
  ```

- reportEvent(eventName, args)

  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)

  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" }`

- getTsl()

  返回TSL字符串，数据格式与云端一致。

  返回值：

  ```
  Promise<Void>
  ```

- getTslConfig()

  > 注意 Link IoT Edge当前版本已不再使用该接口，已使用getTslExtInfo()接口代替，之前遗留的调用不受影响。

  返回TSL配置字符串。

  返回值：

  ```
  Promise<String>
  ```

- getTslExtInfo()

  返回TSL扩展信息字符串。

  返回值：

  ```
  Promise<String>
  ```

- cleanup()

  资源回收接口，您可以使用该接口回收您的资源。

  返回值：

  ```
  Promise<Void>
  ```

- unregister()

  从网关中移除设备。请谨慎使用该接口。

  返回值：

  ```
  Promise<Void>
  ```

getConfig()

获取驱动配置字符串，该配置通常在设备与驱动程序关联时由系统自动生成。

> 说明 该接口与static get()接口Config.get()调用方法的区别在于getConfig()返回配置字符串，Config.get()返回配置对象。

返回值：

```
Promise<String>
```

destroy()

销毁库内部所有资源。通常不再使用此库时调用destroy()接口。

返回值：

```
Promise<Void>
```

1.3. Python版本SDK

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

Python版本开源的SDK源码请见开源的Python库。

安装和使用

1. 您可以通过如下命令来安装SDK。

   ```
   pip3 install lethingaccesssdk
   ```

2. 安装完成后，您可以根据SDK接口进行驱动开发。

   > **注意** 完成驱动开发后，直接运行会提示错误，必须通过物联网平台控制台，将已开发的驱动部署到网关中方可执行。部署驱动到网关的操作请参考驱动开发。

   使用SDK开发驱动的示例代码片段如下所示。

```
# -*- coding: utf-8 -*-
import logging
import time
import lethingaccesssdk
from threading import Timer
# Base on device, User need to implement the getProperties, setProperties and callServi
ce function.
class Temperature_device(lethingaccesssdk.ThingCallback):
    def __init__(self):
        self.temperature = 41
        self.humidity = 80
    def getProperties(self, input_value):
        '''
        Get properties from the physical thing and return the result.
        :param input_value:
        :return:
        '''
        retDict = {
            "temperature": 41,
            "humidity": 80
        }
        return 0, retDict
    def setProperties(self, input_value):
        '''
        Set properties to the physical thing and return the result.
        :param input_value:
        :return:
        '''
        return 0, {}
    def callService(self, name, input_value):
```

```
    def callService(self, name, input_value):
        '''
        Call services on the physical thing and return the result.
        :param name:
        :param input_value:
        :return:
        '''
        return 0, {}
def thing_behavior(client, device):
    while True:
        properties = {"temperature": device.temperature,
                      "humidity": device.humidity}
        client.reportProperties(properties)
        client.reportEvent("high_temperature", {"temperature": 41})
        time.sleep(2)
try:
    thing_config = lethingaccesssdk.Config().getThingInfos()
    for config in thing_config:
        device = Temperature_device()
        client = lethingaccesssdk.ThingAccessClient(config)
        client.registerAndonline(device)
        t = Timer(2, thing_behavior, (client, device))
        t.start()
except Exception as e:
    logging.error(e)
# don't remove this function
def handler(event, context):
    return 'hello world'
```

Config

驱动相关配置信息。

- Config()

  基于当前驱动配置字符串构造新的Config对象。

- getThingInfos()

  返回所有设备相关信息。

  返回值说明如下：

  返回设备用于连接Link IoT Edge的配置信息的封装。

  返回参数

名称	类型	描述
ThingInfo	List	设备信息。

  ThingInfo参数说明

名称	类型	描述
productKey	String	产品唯一标识。

名称	类型	描述
deviceName	String	设备名称。
custom	Object	设备自定义配置。

- getDriverInfo()

  返回驱动相关信息。

  返回值：

  ```
  dict
  ```

ThingCallback

首先根据真实设备，命名一个类（如Demo_device）继承ThingCallback，然后在该类（Demo_device）中实现setProperties、getProperties和callService三个函数。

- setProperties

  设置具体设备属性函数。

  请求参数

名称	类型	描述
properties	Dict	设置属性，取值格式为： `{ "property1": "value1", "property2": "value2" }`

  返回参数

名称	类型	描述
code	Integer	若获取成功则返回0，失败则返回非0错误码。
output	Dict	数据内容自定义，例如： `{ "key1": xxx, "key2": yyy, ... }` 若无返回数据，则值为空 `{}` 。

- getProperties

  获取具体设备属性的函数。

  请求参数

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

返回参数

名称	类型	描述
code	Integer	若获取成功则返回0，失败则返回非0错误码。
output	Dict	返回值，例如： `{ 'property1': xxx, 'property2': yyy, ..}.}`

- callService

  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	Dict	服务入参列表，取值格式为： `{ "key1": "value1", "key2": "value2" }`

  返回参数

名称	类型	描述
code	Integer	若获取成功则返回0，失败则返回非0错误码。

名称	类型	描述
output	Dict	数据内容自定义，例如： `{ "key1": xxx, "key2": yyy, ... }` 若无返回数据，则值为空 `{}` 。

ThingAccessClient(config)

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。

请求参数

名称	类型	描述
config	Dict	包括云端分配的ProductKey和deviceName。 例如， `{ "productKey": "xxx", "deviceName": "yyy" }`

- registerAndOnline(ThingCallback)

  将设备注册到网关中并通知网关上线设备。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。

  请求参数

名称	类型	描述
ThingCallback	Object	设备的callback对象。

- reportProperties(properties)

  主动上报设备属性。

  请求参数

名称	类型	描述

名称	类型	描述

properties	Dict	属性中包含的属性key与value，取值格式为：
		{
		"key1": "value1",
		"key2": "value2"
		}

- reportEvent(eventName, args)

  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	Dict	事件中包含的属性key与value，取值格式为：
		{
		"key1": "value1",
		"key2": "value2"
		}

- getTsl()

  返回TSL字符串，数据格式与云端一致。

  返回值：

  ```
  TSL字符串
  ```

- getTslExtInfo()

  返回TSL扩展信息字符串。

  返回值：

  ```
  TSL扩展信息字符串
  ```

- online()

  通知网关设备上线，该接口一般在设备离线后再次上线时使用。

- offline()

  通知网关设备已离线。

- cleanup()

  资源回收接口，您可以使用该接口回收您的资源。

- unregister()

  移除设备和网关的绑定关系。请谨慎使用该接口。

getConfig()

获取驱动相关配置信息。

返回值为驱动配置信息字符串。

1.4. Java版本SDK

设备接入SDK（Java版本）支持开发者使用Java语言开发设备接入驱动（以下简称驱动）。客户网关环境在安装有Link IoT Edge软件的条件下只要满足Java运行环境即可运行Java驱动。

Java版本SDK的源码以及使用示例，请参见开源Java库。

LedaConfig

- 类名全称：`com.aliyun.linkedge.sdk.LedaConfig`
- 类声明：

```
public class LedaConfig
```

- Java方法说明：

限定符和类型	方法和说明
static String	getDriverConfig() 获取驱动配置。
static String	getDeviceConfig() 获取设备配置。
static String	getTsl(String productKey) 获取productKey（产品唯一标识符）对应的物模型（TSL）。
static String	getTslConfig(String productKey) 获取productKey对应的物模型扩展配置。

LedaDevice

- 类名全称：`com.aliyun.linkedge.sdk.LedaDevice`
- 类声明：

```
public class LedaDevice
```

- 构造函数：

```
LedaDevice(String productKey, String deviceName)
```

- Java方法说明：

限定符和类型	方法和说明
LedaData	getProperties(List<String> propertyNameList) 根据指定的属性名称，获取属性值。使用该方法时，需要实现重载（Overload）。
int	setProperties(HashMap<String, Object> properties) 设置属性名称和属性值。使用该方法时，需要实现重载。
LedaData	callService(String methodName, HashMap<String, Object> params) 执行自定义方法。使用该方法时，需要实现重载。
int	online() 上线设备。
int	offline() 下线设备。
int	reportProperties(HashMap<String, Object> properties) 上报设备属性。
int	reportEvents(String eventName, HashMap<String, Object> outputData) 上报设备事件。

数据类

- 类名全称：`com.aliyun.linkedge.sdk.LedaDevice`
- 类声明：

```
public class LedaData
```

- 构造函数：

```
LedaData(int code, HashMap<String, Object> data)
```

- Java方法说明：

限定符和类型	方法和说明
void	setCode(int code) 设置状态码。
int	getCode() 获取状态码。
void	setData(HashMap<String, Object> data) 设置数据内容。
HashMap<String, Object>	getData() 获取数据内容。

错误码

- 类名全称：`com.aliyun.linkedge.sdk.exception.LedaErrorCode`
- 类声明：

```
public class LedaErrorCode {
    public static final int LE_SUCCESS                              = 0;
    public static final int LE_ERROR_UNKNOWN                        = 100000;
    public static final int LE_ERROR_INVALID_PARAM                  = 100001;
    public static final int LE_ERROR_TIMEOUT                        = 100006;
    public static final int LE_ERROR_PARAM_RANGE_OVERFLOW           = 100007;
    public static final int LE_ERROR_SERVICE_UNREACHABLE            = 100008;
    public static final int LEDA_ERROR_DEVICE_UNREGISTER            = 109000;
    public static final int LEDA_ERROR_DEVICE_OFFLINE               = 109001;
    public static final int LEDA_ERROR_PROPERTY_NOT_EXIST           = 109002;
    public static final int LEDA_ERROR_PROPERTY_READ_ONLY           = 109003;
    public static final int LEDA_ERROR_PROPERTY_WRITE_ONLY          = 109004;
    public static final int LEDA_ERROR_SERVICE_NOT_EXIST            = 109005;
    public static final int LEDA_ERROR_SERVICE_INPUT_PARAM          = 109006;
    public static final int LEDA_ERROR_INVALID_JSON                 = 109007;
    public static final int LEDA_ERROR_INVALID_TYPE                 = 109008;
    private static final String LE_SUCCESS_MSG                      = "Success";
/* 请求成功*/
    private static final String LE_ERROR_INVALID_PARAM_MSG          = "Invalid params";
/* 传入参数为NULL或无效*/
    private static final String LE_ERROR_TIMEOUT_MSG                = "Tiemout";
/* 超时*/
    private static final String LE_ERROR_PARAM_RANGE_OVERFLOW_MSG   = "Param range overfl
ow";              /* 参数范围越界*/
    private static final String LE_ERROR_SERVICE_UNREACHABLE_MSG    = "Service unreachabl
e";               /* 服务不可达*/
    private static final String LEDA_ERROR_DEVICE_UNREGISTER_MSG    = "Device has't regis
ter";             /* 设备未注册*/
    private static final String LEDA_ERROR_DEVICE_OFFLINE_MSG       = "Device has offline
";               /* 设备已下线*/
    private static final String LEDA_ERROR_PROPERTY_NOT_EXIST_MSG   = "Property no exist"
;                /* 属性不存在*/
    private static final String LEDA_ERROR_PROPERTY_READ_ONLY_MSG   = "Property only supp
ort read";        /* 属性只读*/
    private static final String LEDA_ERROR_PROPERTY_WRITE_ONLY_MSG  = "Property only supp
ort write";       /* 属性只写*/
    private static final String LEDA_ERROR_SERVICE_NOT_EXIST_MSG    = "Service no exist";
/* 服务不存在*/
    private static final String LEDA_ERROR_SERVICE_INPUT_PARAM_MSG  = "Service param inva
lid";             /* 服务的输入参数不正确错误码*/
    private static final String LEDA_ERROR_INVALID_JSON_MSG         = "Json format invali
d";               /* JSON格式错误*/
    private static final String LEDA_ERROR_INVALID_TYPE_MSG         = "Param type invalid
";               /* 参数类型错误*/
}
```

- Java方法说明：

限定符和类型	方法和说明
String	getMessage(int code) 获取错误码对应的消息。

2.函数计算SDK

2.1. 简介

边缘函数计算SDK主要包含了设备操作、消息发布、函数调用相关功能的接口。

功能概述

以下介绍函数计算SDK的具体功能。

- 设备操作

  设备操作是IoT场景中最常见的需求，为了方便对设备信息进行读写，边缘函数计算SDK提供了如下四个功能API:

  - 获取设备属性
  - 设置设备属性
  - 调用设备服务
  - 根据标签获取设备列表

- 消息发布

  边缘端的消息发布接口，提供向指定Topic发送消息的能力。您还可以结合设置消息路由，可将消息流转到订阅了该Topic消息的其它函数和将消息发送到云端。

- 函数调用

  函数计算是基于事件驱动的编程模型。事件源可以是设备消息和定时器，也可以是其它函数的调用请求。每一个函数计算不仅以一个独立的程序任务存在，还可以跟普通函数一样被调用，接收调用时传入的参数，并返回处理结果。

安装SDK

边缘函数计算SDK提供Node.js版本和Python版本，并已经集成在Link IoT Edge软件包内，可直接引用。

1. 获取SDK库。
   - Node.js SDK 源码，请参见边缘计算SDK开源Node.js库。
   - Python SDK 源码，请参见边缘计算SDK开源Python库。
2. 引用SDK库。
   - 引用Node.js SDK库。

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

   - 引用Python SDK库。

     ```
     import lecoresdk
     ```

2.2. Nodejs

2.2.1. Publish

调用该接口发布消息到自定义的Topic。

publish(params,callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见下表params参数说明。
callback(err)	function	回调函数。遵循JavaScript标准实践。 • 调用成功，err为null。 • 调用失败，err包含发生的错误信息。

params参数说明

参数	类型	描述
topic	String	接收消息的目标Topic。
payload	String	消息负载。

调用示例

以下示例中定义为每当有外部事件触发时，向 `/hello/world` 这个Topic 发送一条消息。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
exports.handler = function (event, context, callback) {
  var message = {
    topic: '/hello/world',
    payload: 'Hello World.',
  };
  iotData.publish(message, (err, data)=> {
    if (err == null) {
      console.log("-- Publish HelloWorld success.");
    }
    callback(err);
  });
};
```

2.2.2. getThingProperties

调用该接口获取指定设备的某项属性值。

getThingProperties(params, callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见表params参数说明。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。具体请参见表callback参数说明。

params 参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
payload	Array	包含要获取的设备属性的identifier数组。调用成功将返回该属性当前的对应值。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各属性的identifier。

callback参数说明

参数	类型	描述
err	Error	• 调用成功，err为null。 • 调用失败，err包含发生的错误信息。
data	object	返回指定设备属性的值。

调用示例

以下示例获取设备LightDev2的LightSwitch属性值（即开关状态）。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
const getPropertiesParams = {
  productKey: 'a1ZJTVs****', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
  payload: ['LightSwitch']   // The property defined in the Product TSL.
};
/* Promise wrapper for getThingProperties. */
function getThingProperties(params) {
  return new Promise((resolve, reject) => {
    iotData.getThingProperties(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}
exports.handler = function (event, context, callback) {
  getThingProperties(getPropertiesParams).then((data) => {
    console.log(data); // Return value example: { LightSwitch: 0 }
    if (null == data.LightSwitch) {
      console.log("-- [Warn] No property LightSwitch return.");
    } else {
      console.log("-- [Success] LightSwitch = " + data.LightSwitch);
      if (data.LightSwitch == '0') {
        console.log("-- Light is off.");
      } else {
        console.log("-- Light is on.");
      }
    }
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.3. setThingProperties

调用该接口为指定设备设置某项属性值。

setThingProperties(params, callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见表params参数说明。
callback(err)	function	回调函数。遵循JavaScript标准实践。 • 调用成功，err为null。 • 调用失败，err包含发生的错误信息。

params参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
payload	Object	包含属性identifier和要设置的属性值。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各属性的identifier和取值范围。

调用示例

以下示例设置设备LightDev2的LightSwitch属性值为1（即“开”状态）。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
const setPropertiesParams = {
  productKey: 'a1ZJTVs****', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
  payload: {'LightSwitch': '1'} // The property defined in the Product TSL.
};
/* Promise wrapper for setThingProperties. */
function setThingProperties(params) {
  return new Promise((resolve, reject) => {
    iotData.setThingProperties(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}
exports.handler = function (event, context, callback) {
  setThingProperties(setPropertiesParams).then((data) => {
    console.log("-- Set Property Success. Return " + JSON.stringify(data));
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.4. callThingService

调用该接口调用指定设备的某项服务。

callThingService(params, callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见表params参数说明。

参数	类型	描述
callback(err, data)	function	回调函数。遵循JavaScript标准实践。具体请参见表callback参数说明。

params参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
service	String	被调用的服务identifier。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各服务的identifier。
payload	String|Buffer	包含指定服务的输入参数的identifier和值。 可以通过**物模型 TSL**，查看服务的输入参数identifier和取值范围。

callback参数说明

参数	类型	描述
err	Error	• 调用成功，err为null。 • 调用失败，err包含发生的错误信息。
data	object	被调用服务的返回值。

调用示例

以下示例调用设备LightDev2的服务turn。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
const callServiceParams = {
  productKey: 'a1ZJTVs****', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
  service: 'turn',           // The service defined in the Product TSL.
  payload: {'LightSwitch': 0},
};
/* Promise wrapper for callThingService. */
function callThingService(params) {
  return new Promise((resolve, reject) => {
    iotData.callThingService(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}
exports.handler = function (event, context, callback) {
  callThingService(callServiceParams).then((data) => {
    console.log("-- Call service Success. Return " + JSON.stringify(data));
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.5. getThingsWithTags

调用该接口根据标签获取设备列表。

说明

若传入多个标签，仅返回满足所有标签条件的设备列表。

getThingsWithTags(params, callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见表params参数说明。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。具体请参见表callback参数说明。

params参数说明

参数	类型	描述
payload	Array	包含标签信息。一个由*{<标签名>:<标签值>}*组成的数组。

callback参数说明

参数	类型	描述
err	Error	• 调用成功，err为null。 • 调用失败，err包含发生的错误信息。
data	Array	返回满足标签条件的设备信息数组。

调用示例

以下示例查询标签为 `location: master bedroom` 的设备。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
const deviceTags = {
  payload: [{'location': 'master bedroom'}],
};
/* Promise wrapper for getThingsWithTags. */
function getThingsWithTags(params) {
  return new Promise((resolve, reject) => {
    iotData.getThingsWithTags(params, (err, things) => {
      err ? reject(err) : resolve(things);
    });
  });
}
exports.handler = function (event, context, callback) {
  getThingsWithTags(deviceTags).then((things) => {
    console.log(JSON.stringify(things));
    for(var i=0; i<things.length; i++) {
      console.log('-- productKey='+things[i]["productKey"] + ', deviceName='+things[i]["dev
iceName"]);
    }
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.6. invokeFunction

调用该接口，调用指定的函数。

说明

调用函数和发布消息，两者都可以在进程（函数）间传递消息，但区别是：

- 函数调用消息是双向的。调用者发送消息给被调用者后，会收到被调用者处理后返回的消息。
- 发布消息是单向的。发送者不需要被调用者的返回信息。

invokeFunction(params, callback)

参数	类型	描述
params	object	参数对象。需包含的必需参数，请参见表params参数说明。
callback(err, data)	function	回调函数。遵循JavaScript标准实践。具体请参见表callback参数说明。

params 参数说明

参数	类型	描述
serviceName	String	服务名。函数在阿里云函数计算中所属服务的名称。
functionName	String	函数名，在阿里云函数计算中设置的函数名。
invocationType	String	调用类型。 • *Sync*：同步调用。 • *Async*：异步调用。 默认为同步调用。
payload	String|Buffer	参数信息作为函数的输入。

callback参数说明

参数	类型	描述
err	Error	• 调用成功，err为null。 • 调用失败，err包含发生的错误信息。
data	object	被调函数的返回值。

调用示例

- 调用者函数代码示例

  在Invoker中，调用serviceName=EdgeFC， functionName=helloworld的函数。

```
'use strict';
const leSdk = require('linkedge-core-sdk');
const fc = new leSdk.FCClient();
module.exports.handler = function (event, context, callback) {
  var ctx = {
    custom: {
      data: 'Custom context from Invoker',
    },
  };
  var invokerContext = Buffer.from(JSON.stringify(ctx)).toString('base64');
  var invokeParams = {
    serviceName: 'EdgeFC',
    functionName: 'helloworld',
    invocationType: 'Sync',
    invokerContext: invokerContext,
    payload: 'String message from Invoker.',
  };
  fc.invokeFunction(invokeParams, (err, data) => {
    if (err) {
      console.log(err);
    } else {
      console.log(data.payload); // Message from helloworld
    }
    callback(null);
  });
};
```

- 被调用函数代码示例

  以下helloworld函数代码表示被调用函数如何解析调用者传入的参数，以及如何返回结果给调用者。

```
module.exports.handler = function(event, context, callback) {
  console.log(event);   // String message from Invoker.
  console.log(context); // { requestId: '4', invokerContext: {custom: { data: 'Custom dat
a from Invoker' }}}
  console.log(context.invokerContext.custom.data); // Custom data from Invoker
  console.log('-- hello world');
  callback(null, 'Message from helloworld'); // Return the result to Invoker.
};
```

2.2.7. CredentialProviderChain

调用该接口，获取阿里云其它云资源的访问凭据。

CredentialProviderChain().resolvePromise()

该接口返回一个Promise对象，通过该返回内容获取访问阿里云其它云资源的临时凭据，凭据会定期更新（默认为15分钟）。因此当有访问其它云资源请求时，请调用该接口更新凭据。凭据格式参数如下：

参数	类型	描述
accessKeyId	String	访问密钥标识。访问密钥由AccessKeyID和AccessKeySecret组成，用于云服务API请求的身份认证。

参数	类型	描述
accessKeySecret	String	访问密钥。
securityToken	String	安全令牌。

调用示例

本文以访问对象存储OSS云服务为例，描述CredentialProviderChain().resolvePromise()的用法。示例代码如下：

> **说明** 示例代码的详细解读可以参考阿里云OSS服务访问。

```
'use strict';
var leSdk = require('linkedge-core-sdk');
var credChain= new leSdk.CredentialProviderChain();
var OSS = require('./node_modules/ali-oss');
exports.handler = function (event, context, callback) {
  // Retrieves group role credential.
  credChain.resolvePromise()
    .then((cred) => {
      console.log('Access Key Id: %s, Access Key Secret: %s, Security Token: %s',
        cred.accessKeyId, cred.accessKeySecret, cred.securityToken);
      // Constructs a client to interact with OSS cloud service.
      var client = new OSS({
        accessKeyId: cred.accessKeyId,
        accessKeySecret: cred.accessKeySecret,
        stsToken: cred.securityToken,
        region: 'oss-cn-shanghai',
        bucket: 'le-fc-bucket',
      });
      /* Upload local file to cloud. */
      return client.put('fileFromEdge.txt', "/linkedge/run/localFile.txt");
    })
    .then((result) => {
      console.log(result);
      callback(null);
    })
    .catch((err) => {
      console.log(err);
      callback(err);
    });
};
```

2.3. Python

2.3.1. publish

调用该接口发布消息到自定义的Topic。

publish(params)

请求参数

参数	类型	描述
params	dict	参数对象。需包含的必需参数，请参见下表params参数说明。

params参数说明

参数	类型	描述
topic	String	接收消息的目标Topic。
payload	String	消息负载。

返回值

参数	类型	描述
return	dict	调用成功，则返回成功信息；否则返回出错信息。 **说明** 返回的成功信息仅表示接口调用成功，不代表消息发送成功。

调用示例

以下示例中定义每当有外部事件触发时，向 `/topic/hello` 这个Topic 发送一条消息。

```
# -*- coding: utf-8 -*-
import lecoresdk
lesdk = lecoresdk.IoTData()
def handler(event, context):
  pub_params = {"topic": "/topic/hello",
                "payload": "Hello World"}
  lesdk.publish(pub_params)
  print("Publish message to /topic/hello.")
  return 'Hello world'
```

2.3.2. getThingProperties

调用该接口获取指定设备的某项属性值。

getThingProperties(params)

请求参数

参数	类型	描述
params	dict	请求参数对象。需包含的必需参数，请参见下表params参数说明。

params 参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
payload	List	包含要获取的设备属性的identifier数组。调用成功将返回该属性当前的对应值。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各属性的identifier。

返回值

参数	类型	描述
return	dict	调用成功，则返回要获取的设备属性的值；否则返回驱动上报的出错信息。

调用示例

以下示例获取设备LightDev2的LightSwitch属性值（即开关状态）。

```
# -*- coding: utf-8 -*-
import lecoresdk
lesdk = lecoresdk.IoTData()
def handler(event, context):
  get_params = {"productKey": "a1ZJTVsqj2y", # Please replace it with your Product Key.
                "deviceName": "LightDev2",   # Please replace it with your Device Name.
                "payload": ["LightSwitch"]}  # The property defined in the Product TSL.
  res = lesdk.getThingProperties(get_params)
  print(res)
  if 'LightSwitch' in res.keys():
    print("-- [Success] LightSwitch = %s" % res['LightSwitch'])
    if res['LightSwitch'] == '0':
      print("-- Light is off.")
    else:
      print("-- Light is on.")
  else:
    print("-- [Warn] No property LightSwitch return.");
  return 'OK'
```

2.3.3. setThingProperties

调用该接口为指定设备设置某项属性值。

setThingProperties(params)

请求参数

参数	类型	描述
params	dict	请求参数对象。需包含的必需参数，请参见下表params参数说明。

params 参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
payload	dict	包含属性identifier和要设置的属性值。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各属性的identifier和取值范围。

返回值

参数	类型	描述
return	dict	调用成功，则返回true；否则返回驱动上报的出错信息。

调用示例

以下示例设置设备LightDev的LightSwitch属性值为0（即“关”状态）。

```
# -*- coding: utf-8 -*-
import lecoresdk
lesdk = lecoresdk.IoTData()
def handler(event, context):
  set_params = {"productKey": "a1ZJTVs****", # Please replace it with your Product Key.
                "deviceName": "LightDev",    # Please replace it with your Device Name.
                "payload": {"LightSwitch":0}}# The property defined in the Product TSL.
  res = lesdk.setThingProperties(set_params)
  print(res)
  return 'OK'
```

2.3.4. callThingService

调用该接口调用指定设备的某项服务。

callThingService(params)

请求参数

参数	类型	描述
params	dict	请求参数对象。需包含的必需参数，请参见表params参数说明。

params参数说明

参数	类型	描述
productKey	String	设备所属产品的ProductKey，创建产品时，物联网平台为该产品生成的唯一标识。
deviceName	String	设备的DeviceName，创建设备时生成的名称。
service	String	被调用的服务identifier。 在物联网平台控制台，设备所属产品详情页的**功能定义**页，单击**物模型 TSL**，即可查看各服务的identifier。
payload	String|Bytes	包含指定服务的输入参数的identifier和值。 可以通过**物模型 TSL**，查看服务的输入参数identifier和取值范围。

返回值

参数	类型	描述
return	dict	调用成功，则返回被调用服务的返回值；否则返回驱动上报的出错信息。

调用示例

以下示例调用设备LightDev2的turn服务。

```
# -*- coding: utf-8 -*-
import lecoresdk
lesdk = lecoresdk.IoTData()
def handler(event, context):
  get_params = {"productKey": "a1ZJTVs****", # Please replace it with your Product Key.
                "deviceName": "LightDev2",   # Please replace it with your Device Name.
                "service":"turn",            # The service defined in the Product TSL.
                "payload": {"LightSwitch": 1}}
  res = lesdk.callThingService(get_params)
  print(res)
  return 'OK'
```

2.3.5. getThingsWithTags

调用该接口根据标签获取设备列表。

限制说明

若传入多个标签，仅返回满足所有标签条件的设备列表。

getThingsWithTags(params)

请求参数

参数	类型	描述
params	dict	参数对象。需包含的必需参数，请参见表params参数说明。

params参数说明

参数	类型	描述
payload	List	标签信息数组。标签是由*{"key":"value"}*组成的数组。

返回值

参数	类型	描述
return	dict	调用成功，则返回满足条件的设备信息数组；否则返回驱动上报的出错信息。

调用示例

以下示例查询标签为`location: master bedroom`的设备。

```
# -*- coding: utf-8 -*-
import lecoresdk
lesdk = lecoresdk.IoTData()
def handler(event, context):
  get_params = {"payload": [{"location":"master bedroom"}]}
  res = lesdk.getThingsWithTags(get_params)
  print(res)
  for device in res:
    print("device_%s: PK=%s DN=%s" % (res.index(device) + 1, device['productKey'], device['
deviceName']))
  return 'OK'
```

2.3.6. invokeFunction

调用该接口，调用指定的函数。

说明

调用函数和发布消息，两者都可以在进程（函数）间传递消息，但区别是：

- 函数调用消息是双向的。调用者发送消息给被调用者后，会收到被调用者处理后返回的消息。
- 发布消息是单向的。发送者不需要被调用者的返回信息。

invokeFunction(params)

请求参数

参数	类型	描述
params	dict	参数对象。需包含的必需参数，请参见表params参数说明。

params 参数说明

参数	类型	描述
serviceName	String	服务名。函数在阿里云函数计算中所属服务的名称。
functionName	String	函数名，在阿里云函数计算中设置的函数名。
invocationType	String	调用类型。 • *Sync*：同步调用。 • *Async*：异步调用。 默认为同步调用。
payload	String|Bytes	参数信息作为函数的输入。

返回值

参数	类型	描述
return	dict	被调用函数的返回值。

调用示例

- **调用者函数代码示例**

  在Invoker中，调用serviceName=EdgeFC，functionName=helloworld的函数。

```
# -*- coding: utf-8 -*-
import lecoresdk
edgefc = lecoresdk.Client()
def handler(event, context):
  context = {"custom": {"data": "customData"}}
  invokeParams = {
    "serviceName": 'EdgeFC',
    "functionName": 'helloworld',
    "invocationType": 'Sync',
    "invokerContext": context,
    "payload": 'String message from Python Invoker.'
  };
  res = edgefc.invoke_function(invokeParams)
  print(res)
  return 'OK'
```

- **被调用函数代码示例**

  以下helloworld函数代码表示被调用函数如何解析调用者传入的参数，以及如何返回结果给调用者。

```
# -*- coding: utf-8 -*-
import logging
import lecoresdk
def handler(event, context):
  logging.debug(event)
  logging.debug(context)
  return 'hello world'
```

2.3.7. CredentialProviderChain

调用该接口获取云服务访问凭据。

CredentialProviderChain().get_credential()

该接口返回被访问云服务的临时凭据，凭据会定期更新（默认为15分钟）。因此当需要访问云服务时，请调用该接口更新凭据。凭据格式参数如下：

参数	类型	描述
accessKeyId	String	访问密钥标识。访问密钥由AccessKeyID和AccessKeySecret组成，用于云服务API请求的身份认证。
accessKeySecret	String	访问密钥。
securityToken	String	安全令牌。

调用示例

本文以访问OSS云服务为例，描述CredentialProviderChain().get_credential()的用法。示例代码如下：

> **说明** 示例代码中依赖了第三方库oss2，可下载完整代码包putOssFilePy-code.zip，查看完整的代码。

```
# -*- coding: utf-8 -*-
import oss2
import lecoresdk
def handler(event, context):
  cred = lecoresdk.CredentialProviderChain().get_credential()
  auth = oss2.StsAuth(cred['accessKeyId'], cred['accessKeySecret'], cred['securityToken'])
  bucket = oss2.Bucket(auth, 'http://oss-cn-shanghai.aliyuncs.com', 'le-fc-bucket')
  bucket.put_object('fileFromEdgePy.txt', 'Content of object')
  print("-- Put fileFromEdgePy.txt to OSS.")
  return 'OK'
```

3.边缘端OpenAPI

3.1. 概览

边缘端OpenAPI是外部应用访问Link IoT Edge的统一入口。通过OpenAPI外部应用可以获取设备信息、设置设备属性、调用函数计算应用等。

> **说明** 最新版本的边缘端OpenAPI仅适用于v2.9.0及以上版本的Link IoT Edge。若您的Link IoT Edge版本低于v2.9.0，则根据需求选用v2.0版本或v1.0版本OpenAPI。

API列表

- 身份认证类

API名称	API描述
CreateAuthCookie	创建认证Cookie。
DeleteAuthCookie	删除认证Cookie。

- 设备管理类

API名称	API描述
ListThings	列出所有设备及其状态。
SearchThings	按指定过滤条件搜索设备。
SetThingProperties	设置设备属性。
GetThingProperties	获取设备属性。
CallThingServices	调用设备服务。
BulkActions	对设备进行批量操作。

- 物模型管理类

API名称	API描述
GetTSL	获取指定产品的物模型。

- 函数计算类

API名称	API描述
InvokeFunction	调用指定的函数。

- 场景联动类

API名称	API描述
ListScenes	列出所有场景联动规则及其状态。
EnableScene	启用指定的场景联动规则。
DisableScene	停用指定的场景联动规则。

3.2. 调用方式

调用边缘端OpenAPI时，需要遵循请求规范和响应规范。

边缘端OpenAPI对外提供服务的端口号为 `9999` 。为保证传输安全，所有接口仅支持HTTPS协议。

认证授权

目前支持通过Cookie认证方式访问OpenAPI。开发者必须通过CreateAuthCookie接口获取认证Cookie才能进一步调用其它API。具体步骤如下所示。

1. 通过访问如下地址，登录边缘网关控制台，配置API访问权限。

   ```
   https://{ip}:9999
   ```

   其中，*{ip}*为网关所在机器的IP地址。

   > **说明** 登录边缘网关控制台相关操作，请参见登录边缘网关控制台。

2. 调用CreateAuthCookie接口创建认证Cookie。

   CreateAuthCookie接口采用Basic Authentication认证方式，即通过HTTP请求头部 `Authorization` 字段传递认证标识。

   Authorization字段格式如下所示。

   ```
   Authorization: Basic <base64(username:password)>
   ```

   其中，*base64(username:password)*是*<登录边缘网关控制台的用户名:密码>*的Base64编码字符串。

   例如，登录边缘网关控制台的用户名为admin，密码为admin1234，则 `Authorization` 字段内容如下所示。

   ```
   Authorization: Basic YWRtaW46YWRtaW4xMjM0
   ```

3. 从CreateAuthCookie接口的HTTP响应头中，获取 `Set-Cookie` 参数的值。

   `Set-Cookie` 字段格式如下所示。

   ```
   Set-Cookie: token=d71020ad5cb58faf04a6fd81d6df9680582b987eb6d0ce7b33bc3a3e4f******; Max-Age=3600; Path=/
   ```

4. 将CreateAuthCookie接口响应头的 `Set-Cookie` 参数值，当做其它API HTTP请求头的 `Cookie` 参数值。

   HTTP请求头的 `Cookie` 参数格式如下所示。

```
Cookie: token=d71020ad5cb58faf04a6fd81d6df9680582b987eb6d0ce7b33bc3a3e4f******; Max-Age
=3600; Path=/
```

5. 调用除CreateAuthCookie之外的其它OpenAPI时，通过HTTP请求头的 `Cookie` 字段传递认证标识Token，Link IoT Edge会验证该认证标识Token是否具有对应API的访问权限。

响应规范

- 状态码：响应状态码遵循HTTP规范。详细状态码，请参见状态码。
- 响应头：响应头需要包含如下字段默认值。

```
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 07:51:57 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
```

- 响应消息体：响应消息体为JSON格式内容，或者为空。
  - 当调用API成功时，每个API返回的格式和内容都不同，详情请参见具体API接口文档。
  - 当调用API出错时，即响应状态码为非 `2**` 开头的状态码（例如 `302` 、 `400` 等），响应消息体为错误信息内容，格式如下所示。

```
{
  "Code": "string", // What kind of error, such as InvalidParameter.
  "Message": "string" // Detailed description about the error.
}
```

3.3. 快速开始

边缘端OpenAPI支持使用第三方工具Postman进行调试并自动生成调用代码，可节约您编写调用代码的时间。

步骤一：下载安装Postman

Postman是一个API开发协作平台，涵盖API的整个生命周期，即从API设计、开发、调试、测试到生成调用代码。

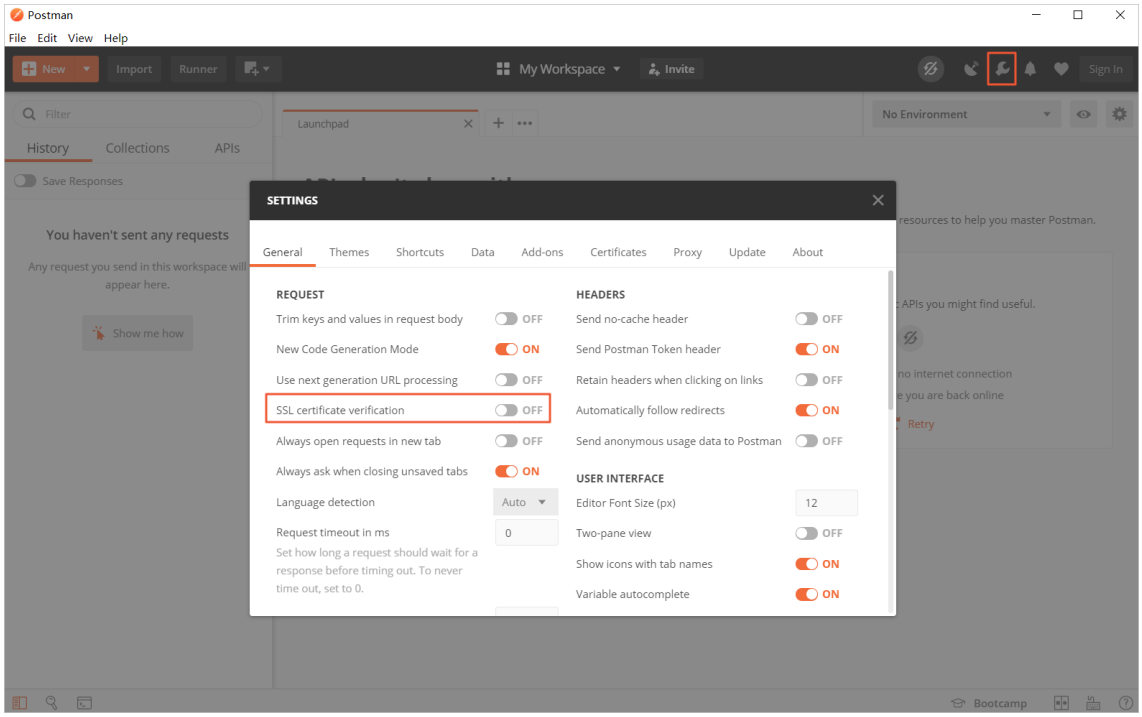
下载并安装Postman。操作详情，请参见Postman官方文档。

> **说明** 本文内容和图片，以Windows 64位系统中下载7.25.0版本Postman为例。

步骤二：设置SSL

Postman默认开启对服务端安全传输协议SSL（Secure Sockets Layer）证书的验证，为保证API正常调用，您可以根据实际情况，选择如下两个方法中的一个，进行SSL设置：

- 关闭SSL验证
  i. 打开Postman程序。
  ii. 在Postman主页面，单击右上角  图标，选择**Settings**。
  iii. 在**SETTINGS**对话框单击**General**页签，将SSL certificate verification开关置为**OFF**。



- 配置CA证书

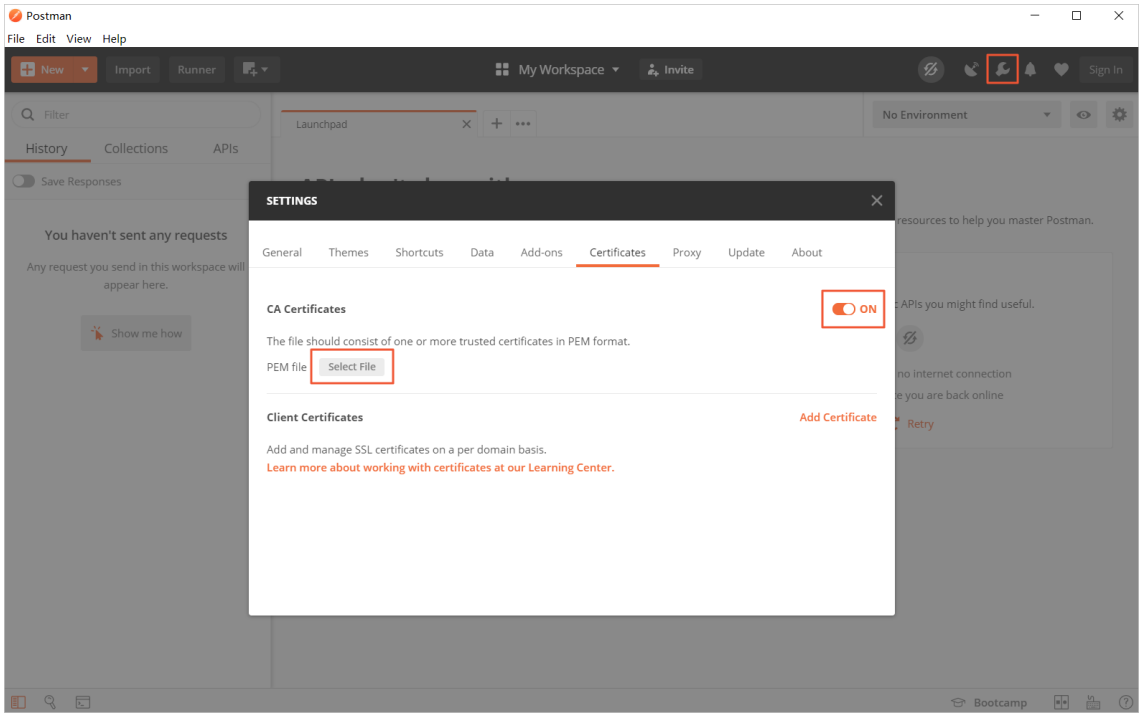
  您可以在Link IoT Edge所在机器上的*/linkedge/gateway/build/bin/console/certs/ca.x509.pem*目录下找到OpenAPI的服务端CA证书，并将该证书下载到本地。关于配置CA证书相关操作，请参见步骤一：设置证书。

  i. 打开Postman程序。

  ii. 在Postman主页面，单击右上角  图标，选择**Settings**。

  iii. 在**SETTINGS**对话框单击**Certificates**页签，将CA Certificates开关置为**ON**。

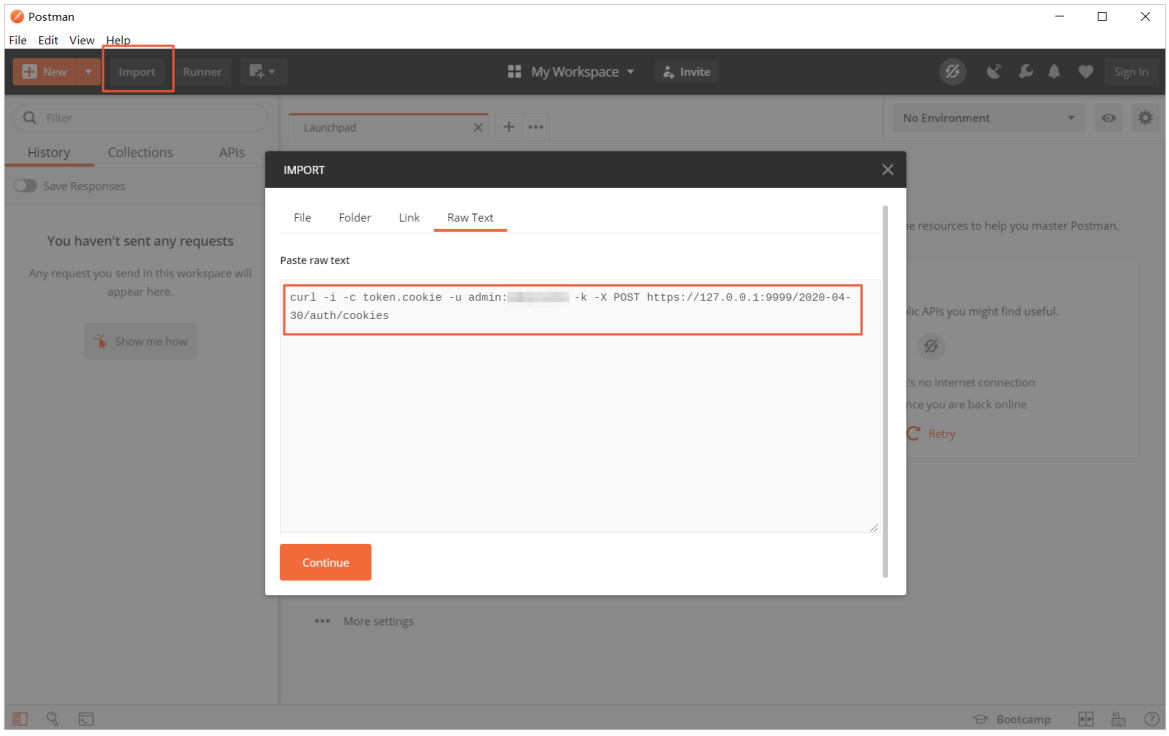
  iv. 单击**Select File**，将已下载到本地的*ca.x509.pem*文件上传到Postman。



步骤三：调试API

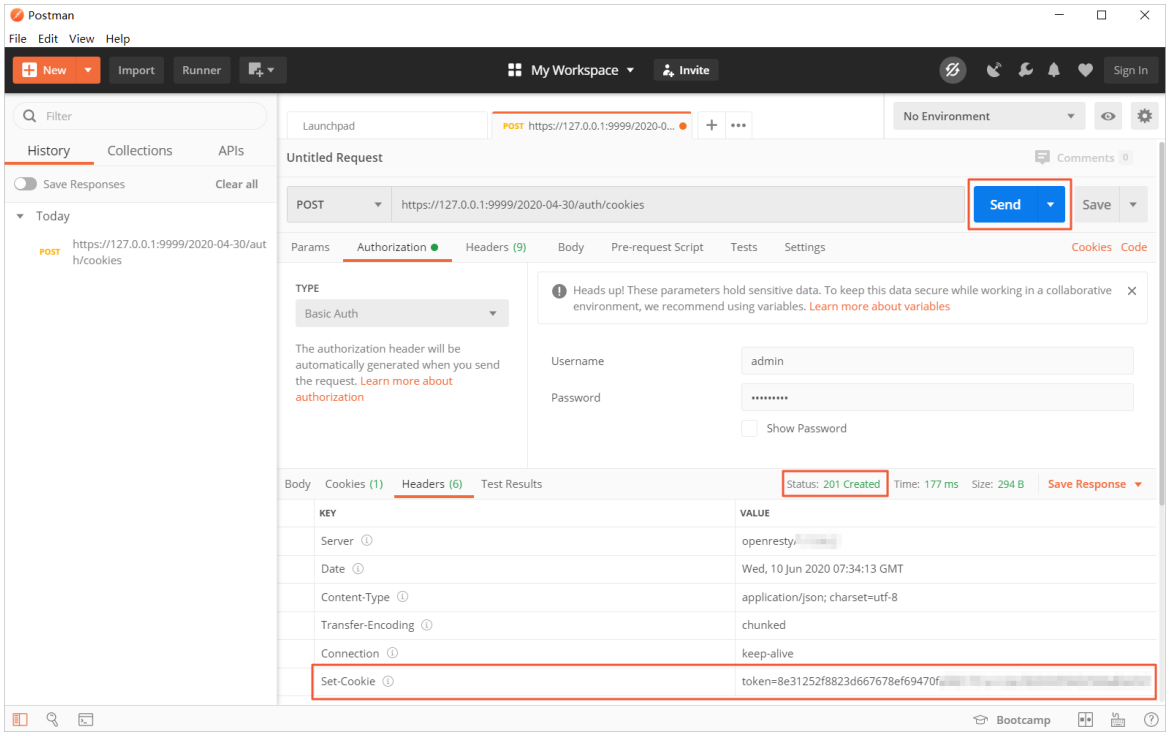
Postman支持从cURL命令导入API请求。

1. 在Postman主页面，单击左上角**Import**。
2. 在弹出的**IMPORT**对话框中单击**Raw Text**页签，输入CreateAuthCookie接口的cURL命令。



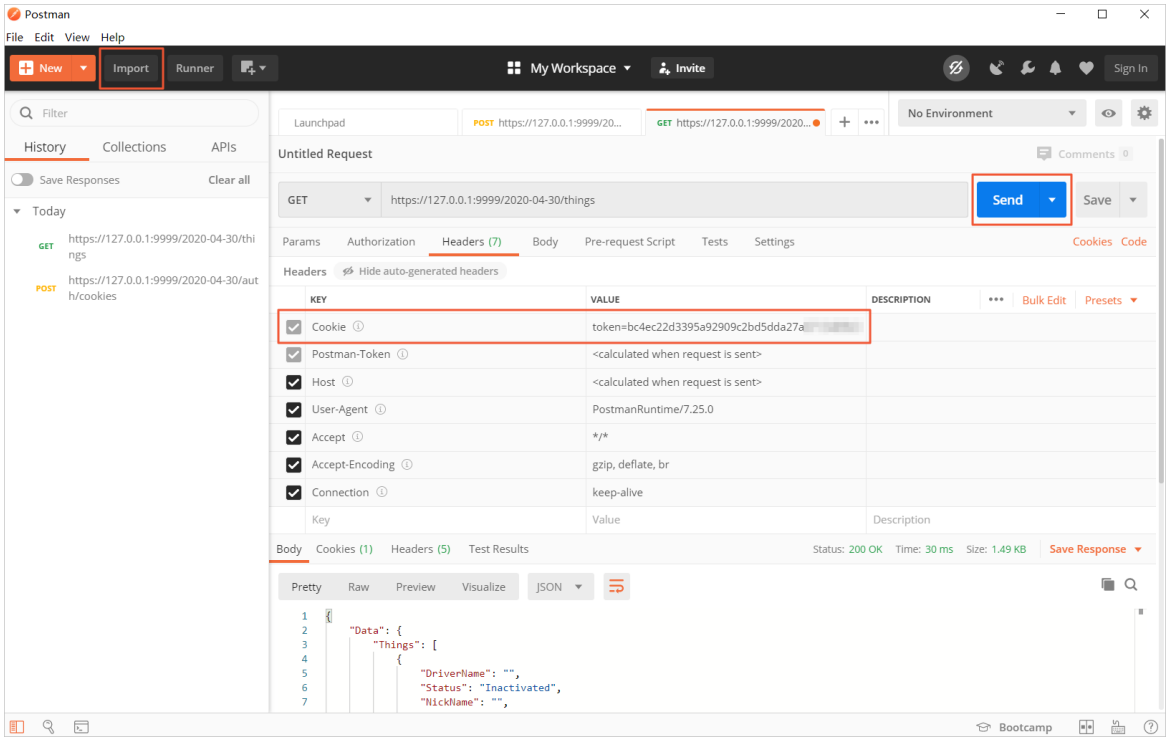
3. 单击**Continue**，系统跳转到**Confirm your import**页面，Postman会解析cURL命令并自动填充请求字段，单击**Import**。
4. 在Postman主页面，单击**Send**发送请求，可查看到返回了201 Created的状态码。

响应消息头（Headers）中的Set-Cookie字段的值，即为认证Cookie，复制该值保存到本地以备后续使用。



5. 获取认证Cookie后，重复步骤1~步骤4，可以调用其它API。

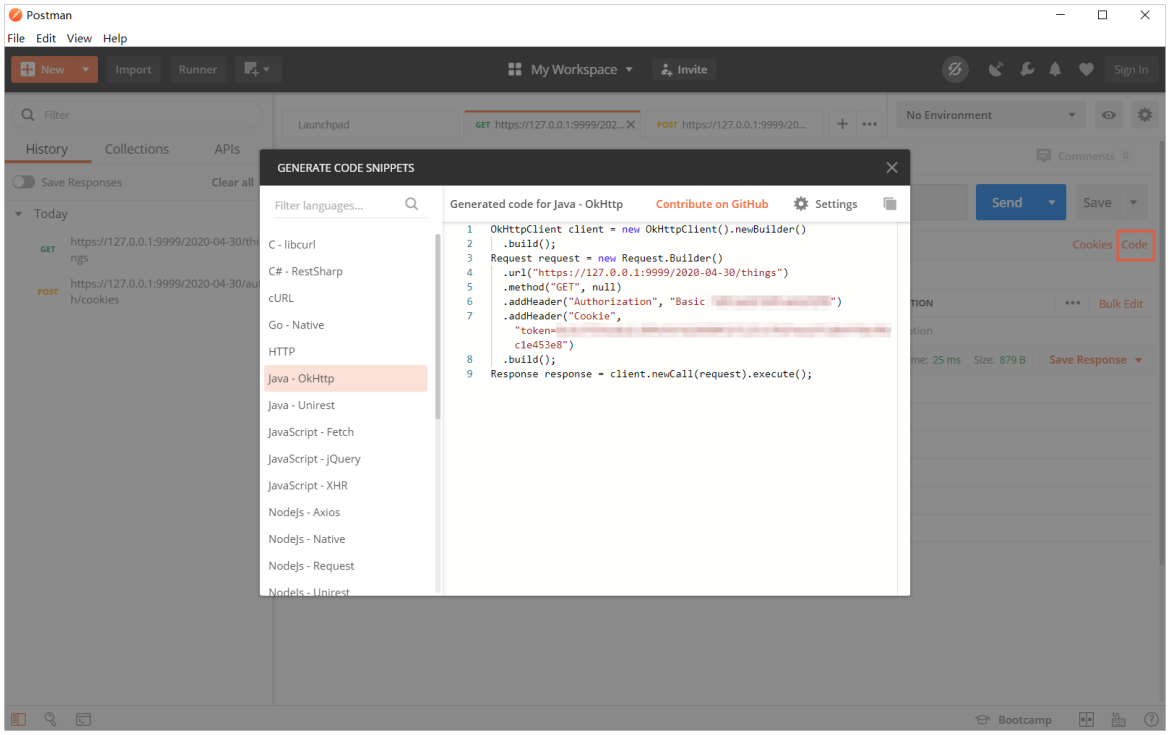
   下图中以调用ListThings接口为例。在Postman主页面，单击左上角**Import**输入ListThings接口的请求cURL命令后，单击**Continue > Import**。可以看到Postman自动将CreateAuthCookie接口响应消息头Set-Cookie字段的内容填入ListThings请求消息头的Cookie字段。然后单击**Send**。



步骤四：生成代码

Postman支持自动生成调用代码。

1. 在Postman主页面已调用成功的API页签右侧单击**Code**。
2. 根据您使用的语言和实际需求，选择语言生成实际的调用代码。



以ListThings接口为例，通过Postman生成的调用代码示例如下：

- Node.js

```
var https = require('https');
var options = {
  'method': 'GET',
  'hostname': '127.0.0.1',
  'port': 9999,
  'path': '/2020-04-30/things',
  'headers': {
    'Cookie': 'token=2c2cbbc36af52216a6dd0bdba38f575e2b94b029d1bf45bb7a377f**********
'
  },
};
var req = https.request(options, function (res) {
  var chunks = [];
  res.on("data", function (chunk) {
    chunks.push(chunk);
  });
  res.on("end", function (chunk) {
    var body = Buffer.concat(chunks);
    console.log(body.toString());
  });
  res.on("error", function (error) {
    console.error(error);
  });
});
req.end();
```

- Python

```
import requests
url = "https://127.0.0.1:9999/2020-04-30/things"
payload = {}
headers = {
  'Cookie': 'token=2c2cbbc36af52216a6dd0bdba38f575e2b94b029d1bf45bb7a377f**********'
}
response = requests.request("GET", url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

- Java

```
OkHttpClient client = new OkHttpClient().newBuilder()
  .build();
Request request = new Request.Builder()
  .url("https://127.0.0.1:9999/2020-04-30/things")
  .method("GET", null)
  .addHeader("Cookie", "token=2c2cbbc36af52216a6dd0bdba38f575e2b94b029d1bf45bb7a377f*
*********")
  .build();
Response response = client.newCall(request).execute();
```

3.4. 状态码

边缘端API的状态码如下表格所示。

HTTP状态码	错误码（Code）	描述
200 OK	无	接口调用成功。
201 Created	无	请求成功并且服务器创建了新的资源。
400 Bad Request	BadRequest	语义有误，当前请求无法被服务器理解，除非进行修改，否则客户端不应该重复提交该请求。 详细的错误信息，请查看本文下方400 Bad Request状态码详细错误信息表格。
401 Unauthorized	Unauthorized	目标资源的身份认证Cookie不正确，需要重新调用CreateAuthCookie接口更新Cookie，否则将无法继续调用其他API。
403 Forbidden	Forbidden	服务器已经理解请求，但拒绝执行请求。需要进行权限校验，确认当前登录用户是否有权限调用请求API。
404 Not Found	NotFound	请求失败，未在服务器上发现请求所希望得到的资源。需要确认请求参数是否正确。
405 Method Not Allowed	MethodNotAllowed	请求行中指定的请求方法不能被用于请求相应的资源。例如，不支持使用POST方法。
500 Internal Server Error	InternalServerError	服务器内部错误。 详细的错误信息，请查看本文下方500 Internal Server Error状态码详细错误信息表格。 **说明** 出现 `500 Internal Server Error` 错误时，请优先根据错误信息提示和日志内容排查问题，若仍无法解决问题，您可以提交工单联系我们获取技术支持。
503 Service Unavailable	ServiceUnavailable	由于临时的服务器维护或服务器过载，因此服务器当前无法处理请求。

400 Bad Request状态码详细错误信息

错误码（Code）	描述
InvalidParameter	请求参数有误。请输入正确的请求参数，然后重试。
MissingParameter	缺少必要的请求参数。请输入正确的请求参数，然后重试。

500 Internal Server Error状态码详细错误信息

错误码（Code）	描述
Cookie.Handler.Create	无法创建身份认证Cookie。

错误码（Code）	描述
Cookie.Set	无法保存身份认证Cookie。
HMAC	认证消息的哈希算法（HMAC）运算失败。
Connection.Create	无法与上游服务建立连接。
Connection.Lost	与上游服务的连接已断开。
Message.Create	无法创建发送到上游服务的消息。
Message.SendAndReply	发送消息到上游服务，或接收上游服务消息异常。
Message.InvalidReply	上游服务响应的格式不正确。
Message.FailedReply	上游服务的响应结果显示，调用接口失败。 当上游服务返回的错误码（Code）不在本行下方 `Service.{name}.{error}` 格式的错误列表中时，会返回该错误码。
Service.NotExist	上游服务不存在。通常上游服务未启动或者退出时，报此错误码。
Service.Data.Get	无法从数据中心获取数据。
Service.Data.Set	无法设置数据到数据中心。
Service.Data.KeyNotExist	无法从数据中心获取数据，因为数据的key不存在。
Service.Thing.Call	调用设备接口失败。详细说明，请参见设备接入状态码。
Service.Function.NotExist	被调函数不存在。
Service.Function.Timeout	函数执行超时。
Service.Scene.NotExist	场景规则不存在。
Service.Scene.NotAtLevel0	禁止执行操作，因为场景联动的运行状态为非Normal模式，例如Sceneless模式等，请更换为Normal模式后重试。

3.5. 身份认证

3.5.1. CreateAuthCookie

使用该接口创建一个新的认证Cookie。

请求语法

```
POST /2020-04-30/auth/cookies
Authorization: Authorization
```

> **注意** 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Authorization	String	是	支持Basic认证。格式为 `Basic base64(username:password)` 。 例如 `Basic Y2h5aW5n********NTY=` 。 **说明** • `username` 为登录边缘网关控制台的用户名。 • `password` 为登录边缘网关控制台的密码。 详情请参见登录边缘网关控制台内容。

返回语法

```
HTTP/1.1 StatusCode
Set-Cookie: Cookie
Content-Type: application/json
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回201表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Cookie	String	授权的认证Cookie，用于进一步的API调用。

完整示例

```
curl -i -c token.cookie -u admin:admin1234 -k -X POST https://127.0.0.1:9999/2020-04-30/auth/cookies
HTTP/1.1 201 Created
Server: openresty/1.13.6.2
Date: Wed, 15 Apr 2020 11:03:05 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: token=e82c2e9b7fe55a42143d89d1d635cd90a8ecef772a9520e3547028ad6a******; Max-Age=3600; Path=/
```

3.5.2. DeleteAuthCookie

使用该接口删除认证Cookie。

请求语法

```
DELETE /2020-04-30/auth/cookies/Token
Cookie: Cookie
```

> 注意 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Token	String	是	认证Cookie的Token。
Cookie	String	是	从CreateAuthCookie中获取的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。

完整示例

```
$ curl -i -b token.cookie -k -X DELETE https://127.0.0.1:9999/2020-04-30/auth/cookies/e82c2
e9b7fe55a42143d89d1d635cd90a8ecef772a9520e3547028ad6a******
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 15 Apr 2020 11:03:40 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
```

3.6. 设备管理

3.6.1. ListThings

使用该接口，列出所有设备及其状态。您也可以传入ProductKey来过滤指定产品下的设备，或者同时传入ProductKey和DeviceName返回某个具体设备。

请求语法

```
GET /2020-04-30/things?productkey=ProductKey&devicename=DeviceName HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	否	设备所属产品唯一标识符。
DeviceName	String	否	设备名。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	接口状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Payload	JSON	已获取的设备信息。

返回Payload格式如下所示。

```
{
    "Data": {
      "Things": [{
        "ProductName": "string",
        "ProductKey": "string",
        "DeviceName": "string",
        "DriverId": "string",
        "DriverName": "string",
        "Tags": [{"string": "string"}],
        "Status": "Inactivated|Failed|Online|Offline",
        "Connected": true|false,
        "ConnectTime": "string"
      }]
    }
  }
```

完整示例

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2020-04-30/things
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 15:18:16 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Things":[{"DriverName":"LightSensor","Status":"Online","NickName":"","DriverId":"
e0bb1b48964e4519b4a52f9b****1dd3","ProductKey":"a1t****n42K","Connected":true,"ConnectTime"
:"1587526047107","DeviceName":"GjCb9LKXgcKXel******","ProductName":"","Tags":[]}]}}
```

3.6.2. SearchThings

使用该接口，按指定过滤条件搜索设备。

请求语法

```
POST /2020-04-30/things/search HTTP/1.1
Cookie: Cookie
Payload
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

参数名称	类型	是否必选	描述
Payload	JSON	是	JSON格式字符串。当前仅支持按标签搜索设备，可设置多个标签，搜索同时满足所有标签的设备。 格式请见表格下方请求Payload格式。

请求Payload格式如下所示。

```
{
  "Tags": {
    "Key1": "value1",
    "Key2": "value2"
    ...
  }
}
```

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。错误码详情，请参见状态码。
Payload	JSON	搜索到的设备信息。

返回Payload格式如下所示。

```
{
    "Data": {
      "Things": [{
        "ProductName": "string",
        "ProductKey": "string",
        "DeviceName": "string",
        "DriverId": "string",
        "DriverName": "string",
        "Tags": [{"Key1": "Value1"}],
        "Status": "Inactivated|Failed|Online|Offline",
        "Connected": true|false,
        "ConnectTime": "string"
      }]
    }
  }
```

完整示例

```
$ curl -i -b token.cookie -d '{"Tags":{"Name":"LightSensor"}}' -k -X POST https://127.0.0.1
:9999/2020-04-30/things/search
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 15:18:16 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Things":[{"DriverName":"LightSensor","Status":"Online","NickName":"","DriverId":"
e0bb1b48964e4519b4a52f9b****1dd3","ProductKey":"a1t****n42K","Connected":true,"ConnectTime"
:"1587526047107","DeviceName":"test","ProductName":"","Tags":{"Name":"LightSensor"}}]}}
```

3.6.3. GetThingProperties

使用该接口获取指定设备的属性值。

请求语法

```
GET /2020-04-30/things/ProductKey/DeviceName/properties?identifiers=Identifier1&identifiers
=Identifier2&... HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Identifiers	String	是	设备的属性标识符列表。最多可检索100条属性值。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Payload	JSON	已获取的设备属性。

返回Payload格式如下所示。

```
{
  "Data": {
      "Properties": {
        "Identifier1": value1,
        "Identifier2": value2,
        "Identifier3": value3,
            ...
      },
      "Timestamp": 1568262117344
    }
}
```

完整示例

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2020-04-30/things/a1W****CGHU/N0hB9tiVW
WZF****KHgs/properties?identifiers=LightSwitch
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 17:01:50 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Timestamp":1587574910136,"Properties":{"LightSwitch":1}}}
```

3.6.4. SetThingProperties

使用该接口为指定的设备设置属性。

请求语法

```
POST /2020-04-30/things/ProductKey/DeviceName/properties HTTP/1.1
Cookie: Cookie
Payload
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	设置设备属性。格式请见表格下方请求Payload格式。

请求Payload格式如下所示。

```
{
  "Properties": {
    "Identifier1": value1,
    "Identifier2": value2,
    "Identifier3": value3,
      ...
  }
}
```

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Payload	JSON	底层属性设置接口返回的内容。

返回Payload格式如下所示。

```
{
  "Data": {
    "Data": string|boolean|number|array|object, // A optional data that returned from the u
nderlying set
    "Timestamp": 1568262117344
  }
}
```

完整示例

```
curl -i -b token.cookie -d '{"Properties":{"LightSwitch":0}}' -k -X POST https://127.0.0.1:
9999/2020-04-30/things/a1W****CGHU/N0hB9tiVWWZF****KHgs/properties
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 17:06:07 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Data":[],"Timestamp":1587575167259}}
```

3.6.5. CallThingServices

使用该接口进行设备的服务调用。

请求语法

```
POST /2020-04-30/things/ProductKey/DeviceName/services HTTP/1.1
Cookie: Cookie
Payload
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	被调用服务的名称和参数。必须与在物联网平台控制台，为设备所属产品自定义产品功能时，设置的服务名称和参数保持一致。格式请见表格下方请求Payload格式。

请求Payload格式如下所示。

```
{
  "Services": [
    {
      "Name": "string",
      "Args": args // Optional arguments for the service.
    }
  ]
}
```

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	接口返回码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Payload	JSON	调用服务的返回信息。

返回Payload格式如下所示。

```
{
  "Data": {
    "Services": [
      {
        "Name": "string",
        "Returns": {
          "Code": "string", // Error code that returned from the underlying call to the thi
ng service.
          "Message": "string", // Error message that returned from the underlying call serv
ices call.
          "Data": string|boolean|number|array|object, // An optional data that returned fro
m the underlying call services call.
        }
      }
    ],
      "Timestamp": 1568262117344
  }
}
```

完整示例

```
$ curl -i -b token.cookie -d '{"Services":[{"Name":"setColor","Args": {"color":"red"}}]}' -
k -X POST https://127.0.0.1:9999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZFMpALK***/service
s
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:17:47 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Services":[{"Name":"setColor","Returns":{"Message":"success","Data":[]}}],"Timest
amp":1572520667899},"Code":200,"Message":"success"}
```

3.6.6. BulkActions

使用该接口批量操作设备。

请求语法

```
POST /2020-04-30/things/bulk HTTP/1.1
Cookie: Cookie
Payload
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	JSON对象，包含批量操作的动作（BulkAction）、操作对象（Things）及其相关参数。格式详情，请参见表格下方Payload格式。

请求Payload格式如下所示。

```
{
  "BulkAction": "string"
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      ... // Parameters for BulkAction
    }
  ]
}
```

Payload参数说明

参数名称	类型	是否必选	描述

参数名称	类型	是否必选	描述
BulkAction	String	是	指定请求的批量操作。 必须是如下指定参数之一： • SetProperites：表示批量设置设备属性。 • GetProperites：表示批量获取设备属性。 • CallServices：表示批量调用设备服务。 • Watch：表示获取设备的某一组指定事件。
Things	Array	是	JSON数组，包含操作对象及其对应的操作参数。格式详情，请参见本文下方GetProperties、SetProperties、CallServices、Watch中请求Payload格式的Things参数。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: ContentType
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
ContentType	String	返回消息（Payload）的类型。取决于批量操作的动作（BulkAction）。
Payload	JSON	批量操作的返回结果。

返回Payload格式如下所示。

```
{
  "Data": {}
}
```

Payload参数说明

参数名称	类型	描述
Data	Object	具体批量操作的返回结果。格式详情，请参见本文下方GetProperties、SetProperties、CallServices、Watch中返回Payload格式的Data参数。

GetProperties

- 请求Payload格式

```
{
  "BulkAction": "GetProperties",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Identifiers": [ // Property Identifiers
        "string",
        "string",
        "string",
        ...
      ]
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Returns": {
          "Code": "string", // Error code
          "Message": "reason for failure", // Error message
          "Data": { // The properties returned from the underlying get properties call.
            "Identifier1": value1,
            "Identifier2": value2,
            "Identifier3": value3
              ...
          }
        },
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
$ curl -i -b token.cookie -d '{"BulkAction":"GetProperties","Things":[{"ProductKey":"a1W*
***CGHU","DeviceName":"N0hB9tiVWWZF****KHgs","Identifiers":["LightSwitch"]}]}' -k -X POST
https://127.0.0.1:9999/2020-04-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 17:41:04 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Returns":{"Data":{"LightSwitch":0}},"DeviceName":"N0hB9tiVWWZF****K
Hgs","ProductKey":"a1W****CGHU"}],"Timestamp":1587577264304}}
```

SetProperties

- 请求Payload格式

```
{
  "BulkAction": "SetProperties",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Properties": {
        "Identifier1": value1,
        "Identifier2": value2,
        "Identifier3": value3,
    ...
      }
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Returns": {
          "Code": "string", // Error code
          "Message": "reason for failure", // Error message
          "Data": string|boolean|number|array|object // A optional data that returned fro
m the underlying set properties call.
        },
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
curl -i -b token.cookie -d '{"BulkAction":"SetProperties","Things":[{"ProductKey":"a1W***
*CGHU","DeviceName":"N0hB9tiVWWZF****KHgs","Properties":{"LightSwitch":1}}]}' -k -X POST
https://127.0.0.1:9999/2020-04-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 17:44:21 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Returns":{"Data":[]},"DeviceName":"N0hB9tiVWWZF****KHgs","ProductKe
y":"a1W****CGHU"}],"Timestamp":1587577461238}}
```

CallServices

- 请求Payload格式

```
{
  "BulkAction": "CallServices",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Services": [
        {
          "Name": "string",
          "Args": args // Optional arguments for the service.
        }
      ]
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Services": [
          {
            "Name": "string",
            "Returns": {
              "Code": "string", // Error code
              "Message": "reason for failure", // Error message
              "Data": string|boolean|number|array|object // A optional data that returned
from the underlying call services call.
            }
          }
        ],
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
curl -i -b token.cookie -d '{"BulkAction":"CallServices","Things":[{"ProductKey":"a1W****
CGHU","DeviceName":"N0hB9tiVWWZF****KHgs","Services":[{"Name":"setColor","Args":{"color":
"red"}}]}]}' -k -X POST https://127.0.0.1:9999//2020-04-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 22 Apr 2020 17:47:02 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Services":[{"Name":"setColor","Returns":{"Data":[]}}],"DeviceName":
"N0hB9tiVWWZF****KHgs","ProductKey":"a1W****CGHU"}],"Timestamp":1587577622555}}
```

Watch

- 请求Payload格式

```
{
  "BulkAction": "Watch",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Watches": "all|properties|events" // default is "all"
    }
  ]
}
```

- 返回Payload格式

  Content-Type：text/plain

- 常规消息

```
{
  "Message": "success",
  "Data": {
    "ProductKey": "string",
    "DeviceName": "string",
    "Event": "string",
    "Args": {}, // Optional arguments along with the event.
    "Timestamp": 1568010749340
  }
}
```

- 心跳消息

```
{
  "Message": "heartbeat",
  "Data": {
    "Timestamp": 1568010749340
  }
}
```

- 完整示例

```
$ curl -b token.cookie -d '{"BulkAction":"Watch","Things":[]}' -k -X POST https://127.0.0
.1:9999//2020-04-30/things/bulk
{"Message":"success","Data":{"Timestamp":1587577766731,"Event":"propertiesChanged","Produ
ctKey":"a1t9b******","Args":{"MeasuredIlluminance":{"time":1587577766729,"value":200}},"D
eviceName":"GjCb9LKXgcKXel******"}}
{"Message":"heartbeat","Data":{"Timestamp":1587577767732}}
{"Message":"success","Data":{"Timestamp":1587577768734,"Event":"propertiesChanged","Produ
ctKey":"a1t9b******","Args":{"MeasuredIlluminance":{"time":1587577768733,"value":100}},"D
eviceName":"GjCb9LKXgcKXel******"}}
```

3.7. 物模型管理

3.7.1. GetTSL

使用该接口获取指定产品的物模型数据。

请求语法

```
GET /2020-04-30/tsls/ProdcutKey HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
Payload	JSON	已获取的产品物模型信息，格式请参见表格下方Payload格式。

返回Payload格式如下所示。

```
{
    "Data": {
      "TSL": {}
    }
  }
```

完整示例

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2020-04-30/tsls/a1t****n42K
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Tue, 21 Apr 2020 13:30:37 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"TSL":{"schema":"https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json","services":[{"desc":"属性设置","method":"thing.service.property.set","identifier":"set","inputData":[],"required":true,"outputData":[],"name":"set","callType":"async"},{"desc":"属性获取","method":"thing.service.property.get","identifier":"get","inputData":["MeasuredIlluminance"],"required":true,"outputData":[{"dataType":{"type":"double","specs":{"max":"65535","unit":"Lux","step":"0.01","min":"0","unitName":"照度"}},"identifier":"MeasuredIlluminance","name":"光照度检测值"}],"name":"get","callType":"async"}],"profile":{"productKey":"a1t****n42K"},"events":[{"desc":"属性上报","method":"thing.event.property.post","identifier":"post","type":"info","outputData":[{"dataType":{"type":"double","specs":{"max":"65535","unit":"Lux","step":"0.01","min":"0","unitName":"照度"}},"identifier":"MeasuredIlluminance","name":"光照度检测值"}],"name":"post","required":true},{"method":"thing.event.LowIlluminance.post","identifier":"LowIlluminance","type":"alert","outputData":[{"dataType":{"type":"double","specs":{"max":"65535","unit":"lm","min":"0","step":"0.01"}},"identifier":"MeasuredIlluminance","name":"光照度检测值"},{"dataType":{"type":"int","specs":{"max":"50","min":"-50","step":"1"}},"identifier":"int32","name":"整型参数"},{"dataType":{"type":"float","specs":{"max":"50.0","min":"-50.0","step":"0.1"}},"identifier":"float","name":"单精度浮点型参数"},{"dataType":{"type":"double","specs":{"max":"50.0","min":"-50.0","step":"0.1"}},"identifier":"double","name":"双精度浮点型参数"},{"dataType":{"type":"enum","specs":{"1":"1","2":"2"}},"identifier":"enum","name":"枚举类型参数"},{"dataType":{"type":"bool","specs":{"1":"1","0":"0"}},"identifier":"bool","name":"布尔类型参数"},{"dataType":{"type":"text","specs":{"length":"2048"}},"identifier":"text","name":"字符串类型参数"},{"dataType":{"type":"date","specs":[]},"identifier":"date","name":"时间型"},{"dataType":{"type":"array","specs":{"size":"10","item":{"type":"int"}}},"identifier":"array","name":"数组类型参数"}],"name":"光照度低告警","required":false}],"properties":[{"accessMode":"r","dataType":{"type":"double","specs":{"max":"65535","unit":"Lux","step":"0.01","min":"0","unitName":"照度"}},"required":true,"name":"光照度检测值","identifier":"MeasuredIlluminance"}]}}}
```

3.8. 函数计算

3.8.1. InvokeFunction

使用该接口调用指定的函数。

请求语法

```
POST /2020-04-30/functions/Function/invocations HTTP/1.1
Cookie: Cookie
Payload
```

> **注意** 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Function	String	是	需要调用的函数信息。有如下两种ARN形式： • 完整的ARN形式：格式为 `acs:fc:<yourRegion>:<yourAccountId>:service:<yourServiceName>:function:<yourFunctionName>` 。 • 部分ARN形式：格式为 `service:<yourServiceName>:function:<yourFunctionName>` 。 **注意** ARN的格式必须符合如下正则表达式模式。 `(^acs:fc:([a-z]{2}-[a-z]+(-\d)?)?:(\d{16})?:)?service:([a-zA-Z][\w-]{0,127}):function:([a-zA-Z][\w-]{0,127})$`
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	Payload的内容将原封不动地传给函数。

返回语法

```
HTTP/1.1 StatusCode
X-Fc-Error-Type: ErrorType
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。
ErrorType	String	在函数执行发生错误时出现的参数。有如下两种值： • Handled：表示函数上报的错误。 • Unhandled：表示由函数计算检测并上报的错误，包括函数处理超时或内存不足等错误。
Payload	JSON	函数返回的内容，或者返回如下Payload格式所示的错误信息。

返回Payload格式如下所示。

```
{
    "Message": "string",
    "StackTrace": []
}
```

完整示例

```
$ curl -i -b token.cookie -k -X POST https://127.0.0.1:9999/2020-04-30/functions/service:helloworld:function:helloworld/invocations
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Tue, 21 Apr 2020 14:37:38 GMT
Content-Type: text/plain; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Hello World
```

3.9. 场景联动

3.9.1. ListScenes

使用该接口列出所有场景联动规则及其状态。

请求语法

```
GET /2020-04-30/scenes HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30`是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。错误码详情，请参见状态码。
Payload	JSON	已获取的场景联动规则信息。

返回Payload格式如下所示。

```
{
    "Data": { // The returned data if success, otherwise null.
      "Scenes": [{
        "Id": "string",
        "Name": "string",
        "Running": true|false
      }]
    }
  }
```

完整示例

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2020-04-30/scenes
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Wed, 15 Apr 2020 08:02:06 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Scenes":[{"Running":true,"Name":"scene_test_types","Id":"d14122632aaf4d1d8e325963
****9528"}]}}
```

3.9.2. EnableScene

使用该接口启用指定的场景联动规则。

请求语法

```
POST /2020-04-30/scenes/Scene/enable HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Scene	String	是	指定场景联动规则的ID。调用ListScenes接口获取场景联动规则ID。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。

完整示例

```
$ curl -i -b token.cookie -k -X POST https://127.0.0.1:9999/2020-04-30/scenes/d14122632aaf4d1d8e325963****9528/enable
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 16 Apr 2020 01:51:27 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
```

3.9.3. DisableScene

使用该接口停用指定的场景联动规则。

请求语法

```
POST /2020-04-30/scenes/Scene/disable HTTP/1.1
Cookie: Cookie
```

> **注意** 其中，`2020-04-30` 是边缘端OpenAPI的版本号，请勿修改。

请求参数

参数名称	类型	是否必选	描述
Scene	String	是	指定场景联动规则的ID。调用ListScenes接口获取场景联动规则ID。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情，请参见状态码。

完整示例

```
$ curl -i -b token.cookie -k -X POST https://127.0.0.1:9999/2020-04-30/scenes/d14122632aaf4
d1d8e325963****9528/disable
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 16 Apr 2020 01:51:08 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
```

3.10. v2.0版本

3.10.1. 概述

外部应用可以通过边缘端OpenAPI访问Link IoT Edge并获取设备信息、设置设备属性、调用函数计算等。

> **说明** 该版本OpenAPI推荐在v2.1.2（含）~v2.9.0（不含）版本的Link IoT Edge中使用；v2.1.2以下版本Link IoT Edge中推荐使用v1.0版本OpenAPI；v2.9.0及以上版本Link IoT Edge中使用最新版本的OpenAPI。

API列表

- 身份认证类

API名称	API描述
CreateAuthCookie	创建认证Cookie。
DeleteAuthCookie	删除认证Cookie。

- 设备管理类

API名称	API描述
ListThings	列出设备及其状态。
SetThingProperties	设置设备属性。
GetThingProperties	获取设备属性。
CallThingServices	调用设备服务。
BulkActions	对设备进行批量操作。

- 函数计算类

API名称	API描述
InvokeFunction	调用指定的函数。

3.10.2. 状态码

边缘端API的状态码如下表格所示。

返回码（code）	返回信息	描述
200	Success	接口调用成功。
201	Created	请求成功并且服务器创建了新的资源。
302	Move temporarily	URL重定向。
400	Bad Request	报该错误码可能有如下两种原因： • 语义有误，当前请求无法被服务器理解，除非进行修改，否则客户端不会重复提交该请求。 • 请求参数有误。
401	Unauthorized	当前请求需要用户验证。即Cookie已过期，需要重新调用CreateAuthCookie接口更新Cookie，否则将无法继续调用其他API。
403	Forbidden	服务器已经理解请求，但拒绝执行请求。需要进行权限校验，确认当前登录用户是否有权限调用请求API。
404	Not Found	请求失败，未在服务器上发现请求所希望得到的资源。需要确认请求的API参数是否正确。
405	Method Not Allowed	请求行中指定的请求方法不能被用于请求相应的资源。例如，不支持使用POST方法。
421	Too Many Connections	从当前客户端所在的IP地址连接到服务器的连接数超过了服务器允许的最大限制。 **说明** 此处的IP地址指的是从服务器上看到的客户端地址，例如用户的网关或者代理服务器地址。
500	Internal Server Error	服务器遇到了一个未曾预料的状况，导致无法完成对请求的处理。通常情况下，服务器的源代码出现错误时报该错误码。
503	Service Unavailable	由于临时的服务器维护或服务器过载，因此服务器当前无法处理请求。

3.10.3. CreateAuthCookie

使用该接口创建一个新的认证Cookie。

请求语法

```
POST /2019-09-30/auth/cookies
Authorization: Authorization
```

请求参数

参数名称	类型	是否必选	描述
Authorization	String	是	支持Basic认证。格式为 `Basic base64(username:password)` 。 例如 `Basic Y2h5aW5n********NTY=` 。 **说明** • `username` 为登录边缘网关控制台的用户名。 • `password` 为登录边缘网关控制台的密码。 详情请参见登录边缘网关控制台内容。

返回语法

```
HTTP/1.1 StatusCode
Set-Cookie: Cookie
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回201表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
Cookie	String	授权的认证Cookie，用于进一步的API调用。
Payload	JSON	返回消息。

返回Payload格式如下所示。

```
{
  "Code": 201,
  "Message": "sucess|reason for failure"
}
```

完整示例

```
$ curl -i -c token.cookie -u admin:admin1234 -k -X POST https://127.0.0.1:9999/2019-09-30/auth/cookies
HTTP/1.1 201 Created
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 07:51:57 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: token=57e22d4f9fb237fcf5c0b59abf621ddeecde1ef740a84fbeb78540******bbe4; Max-Age=3600; Path=/
{"Code":201,"Message":"success"}
```

3.10.4. DeleteAuthCookie

使用该接口删除认证Cookie。

请求语法

```
DELETE /2019-09-30/auth/cookies/Token
Authorization: Authorization
```

请求参数

参数名称	类型	是否必选	描述
Token	String	是	认证Cookie的Token。
Authorization	String	是	基本授权信息。格式为 `Basic base64(username:password)` 。 例如 `Basic Y2h5aW5n********NTY=` 。 **说明** • `username` 为登录边缘网关控制台的用户名。 • `password` 为登录边缘网关控制台的密码。 详情请参见登录边缘网关控制台内容。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
Payload	JSON	返回消息。

返回Payload格式如下所示。

```
{
  "Code": 200,
  "Message": "sucess|reason for failure"
}
```

完整示例

```
$ curl -i -u admin:admin1234 -k -X DELETE https://127.0.0.1:9999/2019-09-30/auth/cookies/57e22d4f9fb237fcf5c0b59abf621ddeecde1ef740a84fbeb78540******bbe4
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 07:57:23 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Code":200,"Message":"success"}
```

3.10.5. ListThings

使用该接口列出设备及其状态。

请求语法

```
GET /2019-09-30/things HTTP/1.1
Cookie: Cookie
```

请求参数

参数名称	类型	是否必选	描述
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	接口状态码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
Payload	JSON	已获取的设备信息。

返回Payload格式如下所示。

```
{
    "Code": number,
    "Message": "success|reason for failure",
    "Data": {
      "Things": [{
        "ProductName": "string",
        "ProductKey": "string",
        "DeviceName": "string",
        "DriverId": "string",
        "DriverName": "string",
        "Tags": [{"string": "string"}],
        "Status": "Inactivated|Failed|Online|Offline",
        "Connected": true|false,
        "ConnectTime": "string"
      }]
    }
  }
```

完整示例

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2019-09-30/things
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 08:23:21 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Things":[{"DriverName":"e0bb1b48964e4519b4a52f9b719e1***","ConnectTime":"15725101
91943","DeviceName":"GjCb9LKXgcKXeluG****","DriverId":"","ProductKey":"","Connected":true,"
ProductName":"","Tags":[],"Status":"Online"}]},"Code":200,"Message":"success"}
```

3.10.6. GetThingProperties

使用该接口获取指定设备的属性值。

请求语法

```
GET /2019-09-30/things/ProductKey/DeviceName/properties?identifiers=Identifier1&identifiers
=Identifier2&... HTTP/1.1
Cookie: Cookie
```

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Identifiers	String	是	设备的属性标识符列表。最多可检索100条属性值。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。错误码详情请参见状态码。
Payload	JSON	已获取的设备属性。

返回Payload格式如下所示。

```
{
    "Code": 200,
    "Message": "sucess|reason for failure",
    "Data": {
        "Properties": {
            "Identifier1": "value1",
            "Identifier2": "value2",
            "Identifier3": "value3"
        },
        "Timestamp": 1568262117344
    }
}
```

完整示例

```
$ curl -b token.cookie -k https://127.0.0.1:9999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZF
MpALK***/properties?identifiers=LightSwitch
{"Data":{"Timestamp":1572512318420,"Properties":{"LightSwitch":1}},"Code":200,"Message":"su
ccess"}
```

3.10.7. SetThingProperties

使用该接口为指定的设备设置属性。

请求语法

```
POST /2019-09-30/things/ProductKey/DeviceName/properties HTTP/1.1
Cookie: Cookie
Payload
```

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	设置设备属性。格式请见表格下方请求Payload格式。

请求Payload格式如下所示。

```
{
  "Properties": {
    "Identifier1": value1,
    "Identifier2": value2,
    "Identifier3": value3
      ...
  }
}
```

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
Payload	JSON	底层属性设置接口返回的内容。

返回Payload格式如下所示。

```
{
  "Code": 200,
  "Message": "sucess|reason for failure",
  "Data": {
    "Data": string|boolean|number|array|object, // A optional data that returned from the u
nderlying set
    "Timestamp": 1568262117344
  }
}
```

完整示例

```
$ curl -b token.cookie -d '{"Properties":{"LightSwitch":0}}' -k -X POST https://127.0.0.1:9
999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZFMpALK***/properties
{"Data":{"Data":[],"Timestamp":1572512468781},"Code":200,"Message":"success"}
```

3.10.8. CallThingServices

使用该接口进行设备的服务调用。

请求语法

```
POST /2019-09-30/things/ProductKey/DeviceName/services HTTP/1.1
Cookie: Cookie
Payload
```

请求参数

参数名称	类型	是否必选	描述
ProductKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
DeviceName	String	是	设备的名称。可从物联网平台控制台获取。
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

参数名称	类型	是否必选	描述
Payload	JSON	是	被调用的服务的名称和参数。必须与在物联网平台控制台，为设备所属产品自定义产品功能时设置的，服务名称和参数保持一致。格式请见表格下方请求Payload格式。

请求Payload格式如下所示。

```
{
  "Services": [
    {
      "Name": "string",
      "Args": args // Optional arguments for the service.
    }
  ]
}
```

返回语法

```
HTTP/1.1 StatusCode
Content-Type: application/json
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	接口返回码。返回200表示成功，返回其它状态码表示失败。错误码详情请参见状态码。
Payload	JSON	调用服务的返回信息。

返回Payload格式如下所示。

```
{
  "Code": 200,
  "Message": "sucess|reason for failure",
  "Data": {
    "Services": [
      {
        "Name": "string",
        "Returns": {
          "Message": "success|reason for failure",
          "Data": string|boolean|number|array|object, // A optional data that returned from
the underlying call services call.
        }
      }
    ],
      "Timestamp": 1568262117344
  }
}
```

完整示例

```
$ curl -i -b token.cookie -d '{"Services":[{"Name":"setColor","Args": {"color":"red"}}]}' -
k -X POST https://127.0.0.1:9999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZFMpALK***/service
s
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:17:47 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Services":[{"Name":"setColor","Returns":{"Message":"success","Data":[]}}],"Timest
amp":1572520667899},"Code":200,"Message":"success"}
```

3.10.9. BulkActions

使用该接口批量操作设备。

请求语法

```
POST /2019-09-30/things/bulk HTTP/1.1
Cookie: Cookie
Payload
```

请求参数

参数名称	类型	是否必选	描述
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。

参数名称	类型	是否必选	描述
Payload	JSON	是	JSON对象，包含批量操作的动作（BulkAction）、操作对象（Things）及其相关参数。格式详情请参见表格下方Payload格式。

请求Payload格式如下所示。

```
{
  "BulkAction": "string"
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      ... // Parameters for BulkAction
    }
  ]
}
```

Payload参数说明

参数名称	类型	是否必选	描述
BulkAction	String	是	指定请求的批量操作。 必须是如下指定参数之一： • SetProperites：表示批量设置设备属性。 • GetProperites：表示批量获取设备属性。 • CallServices：表示批量调用设备服务。 • Watch：表示监听设备的某一组指定事件。
Things	Array	是	JSON数组，包含操作对象及其对应的操作参数。格式详情请参见本文下方GetProperties、SetProperties、CallServices、Watch中请求Payload格式的Things参数。

返回语法

```
HTTP/1.1 StatusCode
Content-Type: ContentType
Payload
```

返回参数

参数名称	类型	描述

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
ContentType	String	返回消息（Payload）的类型。取决于批量操作的动作（BulkAction）。
Payload	JSON	批量操作的返回结果。

```
{
  "Code": number,
  "Message": "success|reason for failure"
  "Data": {}
}
```

Payload参数说明

参数名称	类型	描述
Code	Number	接口返回码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
Message	String	调用成功则返回 `success` ；调用失败则返回失败的原因。
Data	Object	具体批量操作的返回结果。格式详情请参见本文下方GetProperties、SetProperties、CallServices、Watch中返回Payload格式的Data参数。

GetProperties

- 请求Payload格式

```
{
  "BulkAction": "GetProperties",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Identifiers": [ // Property Identifiers
        "string",
        "string",
        "string",
        ...
      ]
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Code": number,
  "Message": "success|reason for failure",
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Returns": {
          "Message": "success|reason for failure",
          "Data": { // The properties returned from the underlying get properties call.
            "Identifier1": value1,
            "Identifier2": value2,
            "Identifier3": value3
              ...
          }
        },
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
$ curl -i -b token.cookie -d '{"BulkAction":"GetProperties","Things":[{"ProductKey":"a1Wa
bAEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Identifiers":["LightSwitch"]}]}' -k -X POST
https://127.0.0.1:9999//2019-09-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:30:40 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Returns":{"Message":"success","Data":{"LightSwitch":1}},"DeviceName
":"N0hB9tiVWWZFMpALK***","ProductKey":"a1WabAEC***"}],"Timestamp":1572521440288},"Code":2
00,"Message":"success"}
```

SetProperties

- 请求Payload格式

```
{
  "BulkAction": "SetProperties",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Properties": {
        "Identifier1": value1,
        "Identifier2": value2,
        "Identifier3": value3
    ...
      }
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Code": number,
  "Message": "success|reason for failure",
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Returns": {
          "Message": "success|reason for failure",
          "Data": string|boolean|number|array|object // An optional data that returned fr
om the underlying set properties call.
        },
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
$ curl -i -b token.cookie -d '{"BulkAction":"SetProperties","Things":[{"ProductKey":"a1Wa
bAEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Properties":{"LightSwitch":1}}]}' -k -X POS
T https://127.0.0.1:9999//2019-09-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:46:53 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Returns":{"Message":"success","Data":[]},"DeviceName":"N0hB9tiVWWZF
MpALK***","ProductKey":"a1WabAEC***"}],"Timestamp":1572522413633},"Code":200,"Message":"s
uccess"}
```

CallServices

- 请求Payload格式

```
{
  "BulkAction": "CallServices",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Services": [
        {
          "Name": "string",
          "Args": args // Optional arguments for the service.
        }
      ]
    }
  ]
}
```

- 返回Payload格式

  Content-Type：application/json

```
{
  "Code": number,
  "Message": "success|reason for failure",
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
        "DeviceName": "string",
        "Services": [
          {
            "Name": "string",
            "Returns": {
              "Message": "success|reason for failure",
              "Data": string|boolean|number|array|object // An optional data that returne
d from the underlying call services call.
            }
          }
        ],
      }
    ],
    "Timestamp": 1568262117344
  }
}
```

- 完整示例

```
$ curl -i -b token.cookie -d '{"BulkAction":"CallServices","Things":[{"ProductKey":"a1Wab
AEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Services":[{"Name":"setColor","Args":{"color
":"red"}}]}]}' -k -X POST https://127.0.0.1:9999//2019-09-30/things/bulk
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:52:13 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
{"Data":{"Results":[{"Services":[{"Name":"setColor","Returns":{"Message":"success","Data"
:[]}}],"DeviceName":"N0hB9tiVWWZFMpALK***","ProductKey":"a1WabAEC***"}],"Timestamp":15725
22733310},"Code":200,"Message":"success"}
```

Watch

- 请求Payload格式

```
{
  "BulkAction": "Watch",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Watches": "all|properties|events" // default is "all"
    }
  ]
}
```

- 返回Payload格式

  Content-Type：text/plain

  - 常规消息

```
{
  "Code": number,
  "Message": "success|reason for failure",
  "Data": {
    "ProductKey": "string",
    "DeviceName": "string",
    "Event": "string",
    "Args": {}, // Optional arguments along with the event.
    "Timestamp": 1568010749340
  }
}
```

  - 心跳消息

```
{
  "Code": 200,
  "Message": "heartbeat",
  "Data": {
  "Timestamp": 1568010749340
  }
}
```

- 完整示例

```
$ curl -b token.cookie -d '{"BulkAction":"Watch","Things":[]}' -k -X POST https://127.0.0
.1:9999//2019-09-30/things/bulk
{"Data":{"Timestamp":1572510704112,"Event":"propertiesChanged","ProductKey":"a1t9b6Nn***"
,"Args":{"MeasuredIlluminance":{"time":1572510704110,"value":400}},"DeviceName":"GjCb9LKX
gcKXeluGJ***"},"Code":200,"Message":"success"}
{"Data":{"Timestamp":1572510706116,"Event":"propertiesChanged","ProductKey":"a1t9b6Nn***"
,"Args":{"MeasuredIlluminance":{"time":1572510706114,"value":300}},"DeviceName":"GjCb9LKX
gcKXeluGJ***"},"Code":200,"Message":"success"}
{"Data":{"Timestamp":1572510706117},"Code":200,"Message":"heartbeat"}
{"Data":{"Timestamp":1572510708119,"Event":"propertiesChanged","ProductKey":"a1t9b6Nn***"
,"Args":{"MeasuredIlluminance":{"time":1572510708118,"value":200}},"DeviceName":"GjCb9LKX
gcKXeluGJ***"},"Code":200,"Message":"success"}
```

3.10.10. InvokeFunction

使用该接口调用指定的函数。

请求语法

```
POST /2019-09-30/functions/Function/invocations HTTP/1.1
Cookie: Cookie
Payload
```

请求参数

参数名称	类型	是否必选	描述
Function	String	是	需要调用的函数信息。有如下两种ARN形式。 • 完整的ARN形式：格式为 `acs:fc:<yourRegion>:<yourAccountId>:service:<yourServiceName>:function:<yourFunctionName>` 。 • 部分ARN形式：格式为 `service:<yourServiceName>:function:<yourFunctionName>` 。 **注意** ARN的格式必须符合如下正则表达式模式。 `(^acs:fc:([a-z]{2}-[a-z]+(-\d)?)?:(\d{16})?:)?service:([a-zA-Z][\w-]{0,127}):function:([a-zA-Z][\w-]{0,127})$`
Cookie	String	是	调用CreateAuthCookie接口创建的认证Cookie。
Payload	JSON	是	Payload的内容将原封不动传给函数。

返回语法

```
HTTP/1.1 StatusCode
X-Fc-Error-Type: ErrorType
Payload
```

返回参数

参数名称	类型	描述
StatusCode	Number	HTTP状态码。返回200表示成功，返回其它状态码表示失败。状态码详情请参见状态码。
ErrorType	String	在函数执行发生错误时出现的参数。有如下两种值。 • Handled：表示函数上报的错误。 • Unhandled：表示由函数计算检测并上报的错误，包括函数处理超时或内存不足等错误。
Payload	JSON	函数返回的内容，或者返回如下Payload格式所示的错误信息。

返回Payload格式如下所示。

```
{
    "Message": "string",
    "StackTrace": []
}
```

完整示例

```
$ curl -i -b token.cookie -k -X POST https://127.0.0.1:9999/2019-09-30/functions/service:he
lloworld:function:helloworld/invocations
HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Tue, 19 Nov 2019 09:25:42 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
HelloWorld
```

3.11. v1.0版本

3.11.1. 概述

物联网边缘计算支持使用边缘端API管理接入到网关的设备，包括获取设备及设备列表、设置设备属性、订阅设备事件等。边缘端API的对外提供服务的端口号为 `9999` ，所有接口均支持HTTPS单向认证。

> **说明** 该版本OpenAPI推荐在v2.1.1及以下版本的Link IoT Edge中使用，v2.1.1以上版本Link IoT Edge中推荐使用更高版本的OpenAPI。

调用方式

边缘端OpenAPI支持HTTP或者HTTPS请求，允许POST方法。

权限校验机制

边缘端OpenAPI采用Basic Authentication登录认证方式，开发者必须知道网关的用户名与密码，并且该用户名具备访问相应API的权限才能完成接口调用。

> **说明** 网关的初始用户名为admin，密码为admin1234，可通过访问边缘网关控制台修改用户名密码和API访问权限。访问边缘网关控制台的操作请参考远程服务访问。

API列表

- 权限校验类

  需要先登录网关，通过认证获取cookie后，才能继续调用其他接口。

API名称	API描述
Login	用户登录认证接口。
Logout	用户取消认证接口。

- 设备管理类

  对设备属性/服务/事件的操作将会直接下发到设备上。

API 名称	API 描述
queryThingCount	获取网关子设备（设备接入网关后称为网关子设备）总数。支持按照设备在线状态进行过滤。
getAuthedDeviceList	获取网关子设备列表。支持按照设备标签、设备在线状态过滤。
setThingProperties	设置设备的属性。支持单次设置设备的多个属性。
getThingProperties	获取设备的属性。支持单次获取设备的多个属性。
invokeThingServices	调用设备服务。
getThingsEventsInfo	订阅设备事件。支持订阅多个设备的多个事件。

3.11.2. 状态码

边缘端API的状态码如下表格所示。

返回码（code）	返回信息	描述
200	Success	接口调用成功。
204	Heartbeats	订阅设备事件的心跳包。

返回码（code）	返回信息	描述
302	Move temporarily	URL重定向。
400	Bad Request	报该错误码可能有如下两种原因： • 语义有误，当前请求无法被服务器理解，除非进行修改，否则客户端不会重复提交该请求。 • 请求参数有误。
401	Unauthorized	当前请求需要用户验证。即cookie已过期，需要重新调用Login接口更新cookie，否则将无法继续调用其他API。
403	Forbidden	服务器已经理解请求，但拒绝执行请求。需要进行权限校验，确认当前登录用户是否有权限调用请求API。
404	Not Found	请求失败，未在服务器上发现请求所希望得到的资源。需要确认请求的API参数是否正确。
405	Method Not Allowed	请求行中指定的请求方法不能被用于请求相应的资源。例如，不支持使用POST方法。
421	Too Many Connections	从当前客户端所在的IP地址连接到服务器的连接数超过了服务器允许的最大限制。 **说明** 此处的IP地址指的是从服务器上看到的客户端地址，例如用户的网关或者代理服务器地址。
500	Internal Server Error	服务器遇到了一个未曾预料的状况，导致无法完成对请求的处理。通常情况下，服务器的源代码出现错误时报该错误码。
503	Service Unavailable	由于临时的服务器维护或服务器过载，因此服务器当前无法处理请求。

3.11.3. Login

POST /auth/login

功能描述

用户登录网关认证的接口。调用该接口可获取接口返回的Cookie，只有获取Cookie后才能进行其他相关API的调用。

说明 对其他所有接口的调用均需要携带本接口返回的Cookie。

请求参数

参数名称	类型	是否必选	描述
UserName	String	是	登录网关的用户名。 **说明** 网关的初始用户名为admin，密码为admin1234。可通过访问边缘网关控制台修改用户名密码和API访问权限。访问边缘网关控制台的操作请参考远程服务访问。
Password	String	是	用户名对应的密码。

返回示例

- 正常返回示例

```
{
    "code": 200,
    "message": "success",
    "data": {
        //UserName: 用户名
        "UserName": string
    }
}
```

- 异常返回示例

```
{
  "code": 400,
  "message": "Invalid username or password",
  "data": {
    "UserName": string
  }
}
```

完整示例

```
# 输入:
curl -c token.cookie -d '{"id":111,"version":"1.0","request":{"apiVer":"0.6"},"params":{"Us
erName":"admin","Password":"admin1234"}}' -k -X POST https://127.0.0.1:9999/auth/login
# 输出:
{
    "code":200,
    "message":"success",
    "data":{
        "UserName":"admin"
    }
}
```

3.11.4. Logout

POST /auth/logout

功能描述

用户登出网关的接口，一旦登出成功，当前session将无法继续调用API。

请求参数

参数名称	类型	是否必选	描述
UserName	String	是	登录网关时使用的用户名。 **说明** 网关的初始用户名为admin，可通过访问边缘网关控制台修改用户名密码和API访问权限。访问边缘网关控制台的操作请参考远程服务访问。

返回示例

- 正常返回示例

```
{
    "code":200,
    "message":"success",
    "data":{
    }
}
```

- 异常返回示例

```
{
    "code":400,
    "message":"cookie is invalid",
    "data":{
    }
}
```

完整示例

```
# 输入:
curl -b token.cookie -d '{"id":111,"version":"1.0","request":{"apiVer":"0.6"},"params":{"Us
erName":"admin"}}' -k -X POST https://127.0.0.1:9999/auth/logout
# 输出:
{
    "code":200,
    "message":"success",
    "data":{
    }
}
```

3.11.5. queryThingCount

POST /thing/device/count

功能描述

根据productKey获取设备的数量。

请求参数

参数名称	类型	是否必选	描述
productKey	String	否	设备所属产品的唯一标识符。可从物联网平台控制台获取。
activeStatus	Intger	否	设备的激活状态。取值如下： • null：忽略状态 • 0：未激活 • 1：激活
onlineStatus	Intger	否	设备的在线状态。取值如下： • null：忽略状态 • 0：离线 • 1：在线

返回示例

- 正常返回示例

```
{
    "code": 200,
    "data": {"deviceCount": 30},
    "message": "success"
}
```

- 异常返回示例

```
{
  "code": 401,
  "data": null,
  "message": "resqust auth error"
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{}}' -k -X POST https://127.0
.0.1:9999/thing/device/count
# 输出
{
    "code":200,
    "message":"success",
    "data":{
        "deviceCount":4
    }
}
```

3.11.6. getAuthedDeviceList

POST /thing/device/list-authed

功能描述

获取已认证设备的列表。

请求参数

参数名称	类型	是否必选	描述
productKey	String	否	设备所属产品的唯一标识符。可从物联网平台控制台获取。
driverName	String	否	分配到设备的驱动名称。取值如下： • 官方驱动名称：Modbus或OPCUA或WebSocket。 • 在物联网平台控制台，**边缘计算 > 驱动管理**中自定义的驱动名称。
DeviceTags	JSON	否	设备标签列表。可从物联网平台控制台获取。
LocalState	String	否	设备本地在线状态。取值如下： • Online：在线 • Offline：离线
CloudState	String	否	设备云端在线状态。取值如下： • Inactive：未激活 • ActivationFailed：激活失败 • Online：在线 • Offline：离线
PageSize	Intger	是	指定返回结果中每页显示的记录数量。如果取值为0， 则返回所有设备列表。

参数名称	类型	是否必选	描述
CurrentPage	Intger	是	指定显示返回结果中的第几页的内容。默认值为 1。
SortMethod	String	否	返回结果中设备的排序方式。以设备创建的时间顺序排序，取值如下： • TimeDesc：倒序 • TimeAsc：正序 默认为时间倒序排序。

返回示例

- 正常返回示例

```
{
    "code": 200,
    "data": {
    "TotalNum": int,
    "PageSize": int,
    "CurrentPage": int, //从1开始计算
    "List": [
          //LocalState: [Inactive, ActivationFailed, Online, Offline], Inactive，未激活；ActivationFailed，激活失败；Online，本地在线；Offline，本地离线
          //CloudState: [Inactive, ActivationFailed, Online, Offline], Inactive，未激活；ActivationFailed，激活失败；Online，云端在线；Offline，云端离线
          //DeviceLocalId：设备注册时传入的设备名称
          //DeviceName：设备注册成功后，分配的云端显示的设备名称（设备证书里的DeviceName）
          {"ProductKey": string, "DeviceName": string, "DriverName": string, "DeviceLocalId": string, "DeviceTags": [string, string, string], "LocalState": string, "CloudState": string, "LastLocalOnlineTime": string, "LastCloudOnlineTime": string, "DeviceLocalId": "abcabcedfg2"},
          {"ProductKey": string, "DeviceName": string, "DriverName": string, "DeviceLocalId": string, "DeviceTags": [string, string, string], "LocalState": string, "CloudState": string, "LastLocalOnlineTime": string, "LastCloudOnlineTime": string, "DeviceLocalId": "abcabcedfg2"},
          {"ProductKey": string, "DeviceName": string, "DriverName": string, "DeviceLocalId": string, "DeviceTags": [string, string, string], "LocalState": string, "CloudState": string, "LastLocalOnlineTime": string, "LastCloudOnlineTime": string, "DeviceLocalId": "abcabcedfg2"}
        ]
    },
    "message": "success"
}
```

- 异常返回示例

```
{
  "code": 403,
  "data": null,
  "message": "device not found"
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{"PageSize":15,"CurrentPage":
1}}' -k -X POST https://127.0.0.1:9999/thing/device/list-authed
# 输出
{
    "code":200,
    "message":"success",
    "data":{
        "List":[
            {
                "LocalState":"Offline",
                "DriverName":"websocket",
                "LastCloudOnlineTime":"1536538951000",
                "DeviceName":"F3eKBL2fLEQxSSh2****",
                "CloudState":"Offline",
                "DeviceTags":[
                ],
                "LastLinkTime":"1536538951000",
                "ProductKey":"a1JJ3QE****",
                "LastLocalOnlineTime":"1536538951000",
                "DeviceLocalId":"abcabcedfg2"
            },
            {
                "LocalState":"Offline",
                "DriverName":"websocket",
                "LastCloudOnlineTime":"1536538950000",
                "DeviceName":"bdzP2VNc6mSaaY3U****",
                "CloudState":"Offline",
                "DeviceTags":[
                ],
                "LastLinkTime":"1536538950000",
                "ProductKey":"a1JJ3QE****",
                "LastLocalOnlineTime":"1536538950000",
                "DeviceLocalId":"abcabcedfg1"
            },
            {
                "LocalState":"Offline",
                "DriverName":"websocket",
                "LastCloudOnlineTime":"1536538490000",
                "DeviceName":"B62SxMDuQ3oWoqAJ****",
                "CloudState":"Offline",
                "DeviceTags":[
                ],
                "LastLinkTime":"1536538490000",
                "ProductKey":"a1JJ3QE****",
                "LastLocalOnlineTime":"1536538490000",
                "DeviceLocalId":"abcabcedfg0"
            },
            {
                "LocalState":"Online",
                "DriverName":"gateway_monitor",
                "LastCloudOnlineTime":"1536344986000",
```

```
                "DeviceName":"openapi_gw****",
                "CloudState":"Online",
                "DeviceTags":[
                ],
                "LastLinkTime":"1536344986000",
                "ProductKey":"b1xONOb****",
                "LastLocalOnlineTime":"1536344986000",
                "DeviceLocalId":"openapi_gw****"
            }
        ],
        "CurrentPage":1,
        "TotalNum":4,
        "PageSize":15
    }
}
```

3.11.7. setThingProperties

POST /thing/device/properties/set

功能描述

设置设备的属性。调用该接口后属性值将直接下发到设备上。

请求参数

参数名称	类型	是否必选	描述
productKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
deviceName	String	是	设备的名称。可从物联网平台控制台获取。
properties	JSON	是	设备的属性列表。格式如下： `{ "cpu_core_number":1, "aaa":2 }`

返回示例

- 正常返回示例

```
{
    "code": 200,
    "data": null
}
```

- 异常返回示例

```
{
  "code": 403,
  "data": null
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{"productKey":"a1VCsnA****","deviceName":"cloud_sync_****","properties":{"cpu_core_number":1,"aaa":2}}}' -k -X POST https://127.0.0.1:9999/thing/device/properties/set
# 输出
{
    "code":200,
    "message":"success",
    "data":null
}
```

3.11.8. getThingProperties

POST /thing/device/property/query

功能描述

获取设备的实时属性值。调用该接口会直接把请求下发到设备上。

请求参数

参数名称	类型	是否必选	描述
productKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
deviceName	String	是	设备的名称。可从物联网平台控制台获取。
propertyIdentifier	JSONArray	是	设备的属性标识符。格式如下： `[ "cpu_core_number", "cpu_usage" ]`

返回示例

- 正常返回示例

```
{
    "code": 200,
    "message": "success",
    "data": [
        {
            "iotId;": "",                          //注意，iotId 在边缘端为空。
            "batchId": "",
            "attribute;": "xxx",
            "group;": "",                          //注意，group 在边缘端为空。
            "value": "xxxx",
            "gmtModified": 1237891329
        }
    ]
}
```

- 异常返回参数示例

```
{
  "code": 401,
  "data": null,
  "message": "device not found"
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{"productKey":"a1acbea****","
deviceName":"RijJCeD63B0DXKQs****","propertyIdentifier":["cpu_core_number","cpu_usage"]}}'
-k -X POST https://127.0.0.1:9999/thing/device/property/query
# 输出
{
    "code":200,
    "message":"success",
    "data":[
        {
            "iotId":"",
            "value":"8",
            "gmtModified":"1552035452",
            "attribute":"cpu_core_number",
            "groud":"",
            "batchId":""
        },
        {
            "iotId":"",
            "value":"16.730524",
            "gmtModified":"1552035452",
            "attribute":"cpu_usage",
            "groud":"",
            "batchId":""
        }
    ]
}
```

3.11.9. invokeThingServices

POST /thing/device/service/invoke

功能描述

设备的服务调用。

请求参数

参数名称	类型	是否必选	描述
productKey	String	是	设备所属产品的唯一标识符。可从物联网平台控制台获取。
inputParams	JSON	是	服务入参。必须与在物联网平台控制台，为设备所属产品自定义产品功能时设置的服务输入参数名称保持一致。
method	String	是	服务方法。 必须与在物联网平台控制台，为设备所属产品定义产品功能时设置的服务标识符保持一致。
deviceName	String	是	设备的名称。可从物联网平台控制台获取。

返回示例

- 正常返回示例

```
{
    "code": 200,
    "data": null,
    "message": "success"
}
```

- 异常返回示例

```
{
  "code": 400,
  "data": null,
  "message": "device not found"
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{"productKey":"a1VCsnA****","
deviceName":"Led****","method":"power_on","inputParams":{"a":"1234abc"}}}' -k -X POST https
://127.0.0.1:9999/thing/device/service/invoke
# 输出
{
    "code":200,
    "message":"success",
    "data":{
        "code":0,
        "data":{
            "rtn":"asdfasdf",
            "uid":"111",
            "code":1
        },
        "message":"ok"
    }
}
```

3.11.10. getThingsEventsInfo

POST /thing/device/events/get

功能描述

订阅设备事件。该接口会保持长连接且不可被重复使用，若有设备事件上报，则在本接口中返回给接口调用者。

> **说明** 该接口支持订阅多个设备的多个事件。

该接口保持长连接时，返回消息的格式为 `$Length\r\n\$content$Length\r\n$content` ，即由消息内容长度的16进制、1个 `\r\n` 、实际消息体组成。

请求参数

参数名称	类型	是否必选	描述
eventInfo	JSON	是	请求参数列表。若取值为 `[]` ，则返回所有设备的所有事件。必须符合JSON数组格式，每组必须包含 `productKey` 、 `deviceName` 和 `eventIdentifier` 三个key。具体格式请见请求示例。

请求示例

```
"eventinfo": [{
    "productKey": "aaa", //设备的productKey，用户可以物联网平台设备管理->设备信息页找到设备的 ProductKey。
    "deviceName": "bbb1", //设备名称。用户可以在物联网平台设备管理一>设备信息页找到设备的DeviceName。
    "eventIdentifier": [//物的事件标识符；非必填，为空返回所有事件；有值则与用户在物联网平台创建产品定义产品功能时录入的自定义事件名称保持一致。
      "aaaa", "bbbb", "ccc"
    ]
  },
  {
    "productKey": "aaa",
    "deviceName": "bbb2",
    "eventIdentifier": [//物的事件标识符；非必填，为空返回所有事件；
      "aaaa", "bbbb", "ccc"
    ]
  },
  {
    "productKey": "aaa",
    "deviceName": "bbb3",
    "eventIdentifier": [//物的事件标识符；非必填，为空返回所有事件；
      "aaaa", "bbbb", "ccc"
    ]
  }
]
```

返回示例

- 正常返回示例

```
{
    "code": 200,
    "message": "success",
    "data": {
        "items": {
            "productKey": string,
            "deviceName": string,
            "eventCode": "Error",
            "iotId": "",                          //注意，iotId 在边缘端为空。
            "eventName": "aaaaa",
            "eventType": "info",
            "eventBody": {
                "ErrorCode": 0
            },
            "batchId": "",                        //注意，batchId 在边缘端为空。
            "timestamp": 1516342985261
        },
        "timestamp": 1516343075699
    }
}
//注意，考虑到这里是长连接，由server主动推给client，开发者需要关注Http header里面的Transfer-Encoding:chunked字段。
```

- 异常返回参数示例

```
{
  "code": 401,
  "data": null,
  "message": "device not found"
}
```

- 心跳返回示例

```
{
    "code":204,
    "message":"heartbeats",
    "data":null
}
```

完整示例

```
# 输入
curl -b token.cookie -d '{"request":{"apiVer":"0.6"},"params":{"eventInfo":[{"productKey":"
a1JJ3QE****","deviceName":"1njlnxNbdHs4EmOh****","eventIdentifier":["abc","edf"]},{"product
Key":"a1JJ3QE****","deviceName":"8sQa2kvLKpWitIFF****","eventIdentifier":["abc","edf"]}]}}'
-k -X POST https://127.0.0.1:9999/thing/device/events/get
# 输出
34
{
    "code":204,
    "message":"heartbeats",
    "data":null
}
34
{
    "code":204,
    "message":"heartbeats",
    "data":null
}
151
{
    "code":200,
    "data":{
        "timestamp":"",
        "items":{
            "productKey":string,
            "deviceName":string,
            "iotId":"",
            "eventBody":{
                "params":{
                    "value":{
                        "timestamp":"88888",
                        "markingTime":66666,
                        "designName":"this is design name"
                    },
                    "time":1535206490271
                }
            }
```

```
            },
            "eventType":"machinestatuschange",
            "batchId":"",
            "eventName":"machinestatuschange",
            "eventCode":"",
            "timestamp":""
        }
    },
    "message":"success"
}
151
{
    "code":200,
    "data":{
        "timestamp":"",
        "items":{
            "productKey":string,
            "deviceName":string,
            "iotId":"",
            "eventBody":{
                "params":{
                    "value":{
                        "timestamp":"88888",
                        "markingTime":66666,
                        "designName":"this is design name"
                    },
                    "time":1535206491073
                }
            },
            "eventType":"machinestatuschange",
            "batchId":"",
            "eventName":"machinestatuschange",
            "eventCode":"",
            "timestamp":""
        }
    },
    "message":"success"
}
151
{
    "code":200,
    "data":{
        "timestamp":"",
        "items":{
            "productKey":string,
            "deviceName":string,
            "iotId":"",
            "eventBody":{
                "params":{
                    "value":{
                        "timestamp":"88888",
                        "markingTime":66666,
                        "designName":"this is design name"
                    },
                    "time":1535206492272
```

```
                }
            },
            "eventType":"machinestatuschange",
            "batchId":"",
            "eventName":"machinestatuschange",
            "eventCode":"",
            "timestamp":""
        }
    },
    "message":"success"
}
151
{
    "code":200,
    "data":{
        "timestamp":"",
        "items":{
            "productKey":string,
            "deviceName":string,
            "iotId":"",
            "eventBody":{
                "params":{
                    "value":{
                        "timestamp":"88888",
                        "markingTime":66666,
                        "designName":"this is design name"
                    },
                    "time":1535206493074
                }
            },
            "eventType":"machinestatuschange",
            "batchId":"",
            "eventName":"machinestatuschange",
            "eventCode":"",
            "timestamp":""
        }
    },
    "message":"success"
}
151
{
    "code":200,
    "data":{
        "timestamp":"",
        "items":{
            "productKey":string,
            "deviceName":string,
            "iotId":"",
            "eventBody":{
                "params":{
                    "value":{
                        "timestamp":"88888",
                        "markingTime":66666,
                        "designName":"this is design name"
```

```
                    },
                    "time":1535206494272
                }
            },
            "eventType":"machinestatuschange",
            "batchId":"",
            "eventName":"machinestatuschange",
            "eventCode":"",
            "timestamp":""
        }
    },
    "message":"success"
}
```