设备接入Link Kit SDK

NodeJS SDK

NodeJS SDK

环境要求与配置

注：目前的JS版本的Link Kit SDK最新版本号为1.2.7

环境配置

安装 Node.js 运行环境，版本需要 >=4.0.0 。

通过 npm 包管理工具安装SDK

将SDK安装到Nodejs项目所在目录

适用于开发者已创建自己的项目，然后集成阿里云的Link Kit SDK：

```
npm install alibabacloud-iot-device-sdk --save
```

将SDK进行全局安装

适用于开发者直接编写了一个js文件（该文件引用了阿里云的Link Kit SDK）、然后直接使用node xxx.js运行该js程序的场景。这种情况需要将SDK进行全局安装，命令如下所示：

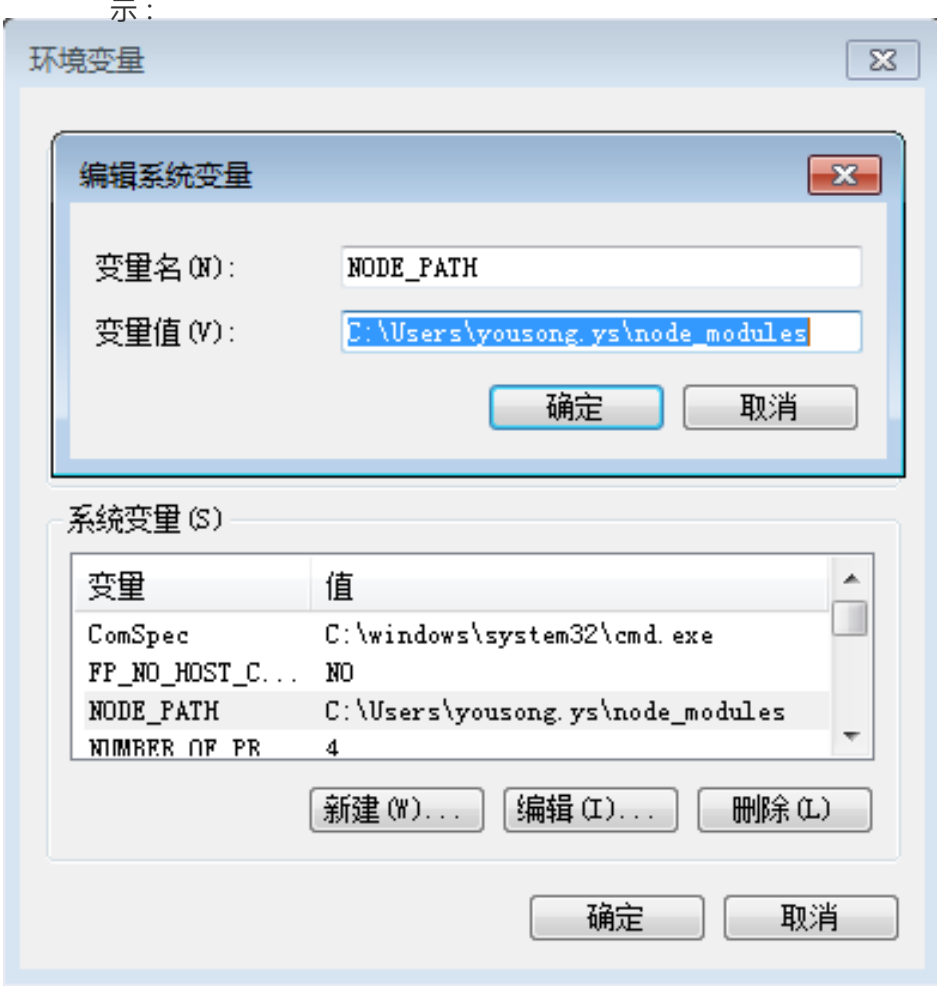
```
npm install -g alibabacloud-iot-device-sdk
```

注：不推荐全局安装的方式，因为扩展性不好。

全局安装后运行程序常见错误：
**

- 如果在**windows**下运行js程序时提示“Error: Cannot find module ‘alibabacloud-iot-device-sdk’ ”，请在环境变量中增加”NODE_PATH”变量，其值设置为SDK安装所在的目录。如下图所

示：



注：上图中的“NODE_PATH”的变量值需要修改为开发者自己设备上SDK安装所在的目录。

- 如果在**Linux**下直接使用node 运行指定的js程序时，提示“Error: Cannot find module ‘alibabacloud-iot-device-sdk’ ”，也是由于没有设置NODE_PATH变量引起，可以运行命令 npm root -g 获取npm的模块安装目录，并将该目录设置到NODE_PATH环境变量并生效（比如重启Linux）后再次使用node加载js程序。

源码下载

目前js版本sdk已经开源，项目地址：https://github.com/aliyun/alibabacloud-iot-device-sdk

如何引用Link Kit SDK

node程序引用SDK

使用SDK提供的API需要在js代码中加入对SDK的依赖，如下所示：

```
const iot = require('alibabacloud-iot-device-sdk');
```

浏览器、微信小程序、支付宝小程序集成SDK

```
const iot = require('./dist/alibabacloud-iot-device-sdk.js'); //引用路径以实际为准
```

JS版本下载地址
github下载：https://github.com/aliyun/alibabacloud-iot-device-sdk/tree/master/dist
cdn下载：
alibabacloud-iot-device-sdk.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.js
alibabacloud-iot-device-sdk.min.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.min.js

版本变更

1.2.7

- 修改了事件上报、属性上报中给出参数的定义
- 增加onProps对属性设置进行处理
- 包名的修改，从 aliyun-iot-device-sdk 正式改名为 alibabacloud-iot-device-sdk
- 增加对微信小程序、支付宝小程序、浏览器的支持
- 增加onService中reply函数,并支持同步和异步调用
- 增加onConfig方法用于订阅云端远程配置更新
- 增加部分功能的example
- 重写了网关子设备subdevice的实现

1.0.1

- 增加设备标签上报功能
- 增加删除标签功能
- 增加了设备动态注册功能
- 增加设备影子相关功能
- 增加获取设备配置功能

认证与连接

设备认证

设备的身份认证支持两种方法，不同方法在填写不同信息。

- 若使用一机一密认证方式，设备上需要烧写product_key、device_name和device_secret
- 若使用一型一密认证方式，设备上需要烧写product_key、和product_secret，并且厂家需要为每个设备设置一个唯一标识（比如SN、MAC地址），这个唯一标识将被用来作为device_name使用，由于product_key和product_secret对于一个产品来说是固定的数值，烧写的时候是固定的，因此称为一型一密。

注：

阿里云IoT的一型一密，是通过一个动态注册过程去获取设备的device_secret，然后再使用product_key、device_name、device_secret去连接阿里云物联网平台并对设备进行认证，本质上仍然是一机一密

动态注册过程只能执行一次，也即当设备获取到device_secret之后，设备需要将其存入NVRAM/FLASH，当重启或者再次连接阿里云物联网平台时，设备需要判断如果设备已经获取到了device_secret，那么不需要再次通过动态注册去获取device_secret，而是直接使用product_key、device_name、device_secret去连接阿里云物联网平台

如果一个设备已经成功的通过动态注册过程获取到了device_secret，再次调用动态注册接口将会返回错误

认证与连接API描述

API原型	**iot.device(options)**
功能描述	创建一个设备实例，并连接阿里云IoT
入参	options： - productKey (String) - deviceName (String) - deviceSecret (String) - region (String) ：阿里云 region，默认值:cn-shanghai - keepalive (int) ：心跳报文时间间隔,默认值60秒 - clean (bool) ：是否清除连接session设置,默认值false
返回值	MQTT Client连接实例
Event	

	- connect：当连接到云端成功时触发 - offline：当连接断开时触发 - message：当接收到来自云端消息时触发 - error：当发生错误时触发，比如PK、DN、DS有误导致连接失败时

使用示例

一机一密设置

在创建实例时输入设备的三元组信息，下面是代码实例：

```
// node引入包名
const iot = require('alibabacloud-iot-device-sdk');

//创建iot.device对象将会发起到阿里云IoT的连接
const device = iot.device({
productKey: '<productKey>', //将<productKey>修改为实际产品的ProductKey
deviceName: '<deviceName>',//将<deviceName>修改为实际设备的DeviceName
deviceSecret: '<deviceSecret>',//将<deviceSecret>修改为实际设备的DeviceSecret
});


//监听connect事件
device.on('connect', () => {
//将<productKey>  <deviceName>修改为实际值
device.subscribe('/<productKey>/<deviceName>/get');
console.log('connect successfully!');
device.publish('/<productKey>/<deviceName>/update', 'hello world!');
});

//监听message事件
device.on('message', (topic, payload) => {
console.log(topic, payload.toString());
});
```

注：

- 如果设备异常断开，程序会自动尝试与云端建立连接
- keep alive默认值是60，设置时不能小于60

设置云端站点 region

阿里云IoT在多个国家与地区部署了服务器，默认地域为上海。厂商可设置设备需要连接的地域，在创建device时将文档中指定的**Region ID** 填入regionId即可，下面的代码示例指定地域为 ap-northeast-1（东京站点）：

```
var device = iot.device({
productKey: '<productKey>',
deviceName: '<deviceName>',
deviceSecret: '<deviceSecret>',
regionId: 'ap-northeast-1'
});

device.on('connect', () => {
console.log('connect successfully!');
});
```

注：region需要与产品在IoT平台上创建产品时所在的region保持一致。

一型一密设置

若产品被设置为一型一密，设备可以通过iot.register()去获取设备的device_secret，示例代码如下：

```
const params = {
productKey:"xxxxxx",
productSecret:"xxxxxx",
deviceName:"xxxxxx"
}

var device;

iot.register(params,(res)=>{
console.log("register:",res);
if(res.code == '200'){
// res.data.deviceSecret 是云端反馈的设备密钥，请妥善保存该密钥，
// 如果设备使用三元组连接物联网平台成功，再次使用本函数去获取DeviceSecret将会失败

//创建设备对象去连接阿里云IoT
device = iot.device({
productKey: '<productKey>',
deviceName: '<deviceName>',
deviceSecret: res.data.deviceSecret, //res.data.deviceSecret 是云端反馈的设备密钥
});
}
})
```

Git中的示例文件：https://github.com/aliyun/alibabacloud-iot-device-sdk/blob/master/examples/one_model_one_secret.js

动态注册时res.code可能的数值列表：

数值	含义说明
200	成功获取DeviceSecret
5005	无效产品，设备提供的ProductKey有误
6100	无效设备，云端查找不到设备提供的deviceName对应的设备

6288	产品不支持动态注册，请在IoT平台为产品打开动态注册功能
6289	设备已经激活，云端拒绝提供DeviceSecret
6600	校验错误，设备提供的ProductSecret有误

断开与云端的连接

如果希望断开与云端的连接，可以调用end函数来断开云端的连接：

```
const iot = require('alibabacloud-iot-device-sdk');

const device = iot.device({
productKey: '<productKey>',
deviceName: '<deviceName>',
deviceSecret: '<deviceSecret>',
});

...

/*Disconnect from aliyun IoT platform*/
device.end();
```

基于MQTT Topic通信

SDK提供了与云端长链接的基础能力接口，用户可以直接使用这些接口完成自定义 Topic 相关的功能。提供的基础能力包括：发布、订阅、取消订阅。

数据上报

API原型	**device#publish(topic, message, [options], [callback])**
功能描述	向指定的topic发送一个消息
参数描述	- topic (String)：topic值 - message :需要发送的消息, 数据格式为 Buffer or String - options: - qos QoS level, 默认值为 0 - callback - function (err), fired when the QoS handling completes, or at the next tick if QoS

	0. An error occurs if client is disconnecting.

代码示例：（请确保产品具有该topic，并且该topic的操作权限为“发布”）

```
//To publish a message with QoS 0
device.publish('/<productKey>/<deviceName>/user/update', 'hello world!');
//To publish a message with QoS 1
device.publish('/<productKey>/<deviceName>/user/update', 'hello world!',{qos:1});
//To publish a Buffer
device.publish('/<productKey>/<deviceName>/user/update', new Buffer([0,1,2,3,4]));
```

消息订阅

API原型	**device#subscribe(topic/topic array/topic object, [options], [callback])**
功能描述	订阅指定topic的消息
参数描述	- topic(string)：某个topic的值，或者指向一个topic数组 - options: - qos QoS level, 默认值为 0 - callback - function (err,granted)：收到suback时调用，其中 - err：error描述 - granted：是{topic, qos}的数组

代码示例：（请确保产品具有相应的topic，并且topic具有“订阅”权限）

```
//订阅指定topic
device.subscribe('/<productKey>/<deviceName>/user/get');
//device.subscribe('/<productKey>/<deviceName>/user/get',{qos:1});

//接收到数据时将topic以及消息打印出来
device.on('message', (topic, payload) => {
console.log(topic, payload.toString());
});
```

参考代码：https://github.com/aliyun/alibabacloud-iot-device-sdk/blob/master/examples/origin.js

取消消息订阅

API原型	**device#unsubscribe(topic/topic array, [callback])**
功能描述	订阅指定topic的消息
参数描述	

	- topic(string)：某个topic的值，或者指向一个topic数组 - callback - function (err)：收到unsuback时调用

代码示例：

```
device.unsubscribe('/<productKey>/<deviceName>/user/get')
```

物模型开发

物模型的开发方式让设备不用关心如何去订阅MQTT topic，而是调用物模型相关的接口来实现属性上报、服务监听、事件上报。

设备属性上报

API原型	**device#postProps(params, [callback])**
功能描述	上报属性
参数描述	- params 属性参数，Object 类型 - callback - res 服务端 reply 消息内容

示例代码：

```
// 上报设备属性
device.postProps({
LightSwitch: 0
}, (res) => {
console.log(res);
});
```

上面的示例代码用于上报一个名为LightSwitch的属性，其值为0，开发者也可参见完整参考代码。

属性设置

调用device.onProps()监听云端下发的属性设置，

API原型	**device#onProps(function(cmd))**

功能描述	监听来自云端的属性设置
参数描述	- function:收到命令时调用的回调函数 - cmd 服务端下发的命令

下面是收到消息的一个示例(cmd的内容)：

```
{
method: 'thing.service.property.set',
id: '802031359',
params: { LightSwitch: 1 },
version: '1.0.0'
}
```

下面是对一个灯的开关属性进行设置时的示范处理代码：

```
// 监听云端设置属性服务消息，示例代码为一个灯
device.onProps((cmd)=>{
console.log('>>>onProps',cmd); //打印完整的属性设置消息
for(var key in cmd.params){
if(key=='LightSwitch'){ //判断是否设置的是LightSwitch属性
console.log('set property ',key);
//示例代码将云端设置的属性在本地进行保存，实际产品开发时需要修改为去将灯打开或者关闭
lightState = cmd.params.LightSwitch;
//本地设置完毕之后，将更新后的状态报告给云端。
//注意：云端下发命令后，云端属性的值并不会改变，云端需要等待来自设备端的属性上报
device.postProps({'LightSwitch': lightState});
}
}
})
```

监听云端下发的服务调用消息

API原型	**device#onService(seviceIdentifier, [callback])**
功能描述	监听服务设置
参数描述	- serviceIdentifier 服务ID，string类型 - callback

```
res 服务端返回参数<br />     reply 响应服务的函数，可以使用同步可以异步方式响应 |
```

下面是服务调用进行处理的代码示例：

```
//示例服务是一个加法器，云端服务调用时给出x和y，返回x和y的和
```

```
function addFunc(x,y){
let err;
if(x==undefined || y==undefined){
err = 'x or y invail value';
return {err,code:10001} //输入参数错误时的格式封装
}
//注意返回的是一个JSON对象，数据结果封装在data中
return {
data:{
z:x+y //z是服务定义中的输出参数
},
code:200
}
}

// subscribe add_async service,产品上定义了add_async的服务
device.onService('add_async', function (res,reply) {
console.log('add_async called,res:',res);
const { params:{x,y}={}} = res; //获取服务参数
const result = addFunc(x,y); //调用addFunc，在该函数中对数据进行编码
console.log('result',result);
reply(result); //返回处理结果
});
```

事件上报

API原型	**device#postEvent(eventIdentifier, params, [callback])**
功能描述	上报事件
参数描述	- eventIdentifier：事件ID，String 类型 - params 事件参数，Object 类型 - callback - err 错误，比如超时 - res 服务端 reply 消息内容

上报 id 为 eventIdentifier1 的事件示例代码：

```
device.postEvent('eventIdentifier1', {
//key1是事件'eventIdentifier1的参数'
key1: 'value1'
});
```

点击此处查看完整代码示例。

标签

功能介绍

物联网平台的设备标签是给设备添加自定义的标识。您可以使用标签功能来灵活管理产品、设备和分组。

设备标签的结构为键值对, Key:Value。

您可以根据设备的特性为设备添加特有的标签，方便对设备进行管理。例如，为房间 201 的智能电表定义一个标签为room:201。

关于标签的更多介绍详见物联网平台标签中的描述。

SDK使用

更新标签

API原型	device#postTags(params, [callback])
函数描述	上报或更新设备标签
参数描述	- params 属性对象数组，array 类型，内容格式示例 [{attrKey:’ xxx’ ,attrValue:’ xxx’ },{}...] - callback - res 服务端 reply 消息内容

下面的示例代码向云端添加一个名为“Temperature”的标签，其值为“36.8”：

```
const tags = [
{
"attrKey": "Temperature",
"attrValue": "36.8"
}
]
device.postTags(
tags,
(res) => {
console.log(`add tag ok res:${res.id}`);
done()
}
);
```

注：设备可以向云端添加多个标签

删除标签

API原型	device#deleteTags(tags)
函数描述	删除设备标签
参数描述	- tags 属性参数，array 类型，内容格式 [‘string’,’ string’,....]，string 内外为tag的标签名称

代码示例：

```
device.deleteTags(['tagA','tagB']);
```

上面的示例代码删除名为“tagA”和“tagB”的两个标签。

开发者也可访问完整**代码示例**了解相关用法。

设备影子

功能介绍

设备影子是一个 JSON 文档，用于存储设备上报的状态、应用程序期望状态信息。

> 每个设备有且只有一个设备影子，设备可以通过MQTT获取和设置设备影子来同步设备与物联网平台上存储的数据，该同步可以是云端的影子同步给设备，也可以是设备同步给云端的影子。
>
> 应用程序通过物联网平台的SDK获取和设置设备影子，获取设备最新状态或者下发期望状态给设备。

具体影子的详细介绍见物联网平台的**设备影子**章节

API说明

监听影子变更

API原型	iot.device#onShadow(callback)
功能说明	监听设备影子的变化，不论是设备端主动更新影子、设备端获取影子都会被调用
参数说明	- callback：回调函数

代码示例：

```
device.onShadow((res) => {

console.log('获取最新设备影子,%o', res);

})
```

用户需要分析返回的内容判断是更新成功、失败、还是云端主动向设备推送设备影子等情况，下面将会分情况进行说明

设备端更新云端影子

API原型	iot.device#postShadow(params)
功能说明	上报设备影子
参数说明	- params：影子的内容，为一个JSON对象

代码示例：

```
device.postShadow({
"a": "avalue"
});
```

设备调用postShadown()之后，device.onShadow()将会被调用，若影子更新成功传递给onShadow()的res内容如下所示：

```
{
"method": "reply",
"payload": {
"status": "success",
"version": 1
},
"timestamp": 1469564576
}
```

用户可以根据method的数值“reply”得知res的数值是云端对更新的响应，根据“status”为“success”得知影子更新成功；若影子更新失败，则传递给onShadow()的res内容如下所示：

```
{
"method": "reply",
"payload": {
"status": "error",
"content": {
"errorcode": "${errorcode}",
"errormessage": "${errormessage}"
}
},
"timestamp": 1469564576
}
```

用户可以根据“status”的值“error”得知更新影子失败，errorcode的数值定义如下：

errorCode	errorMessage
400	不正确的JSON格式
401	影子JSON缺少method信息
402	影子JSON缺少state字段
403	影子JSON version不是数字
404	影子JSON缺少reported字段
405	影子JSON reported属性字段为空
406	影子JSON method是无效的方法
407	影子内容为空
408	影子reported属性个数超过128个
409	影子版本冲突
500	服务端处理异常

查询云端影子数据

API原型	iot.device#getShadow()
功能说明	从物联网平台获取最新的影子数据
参数说明	无

代码示例：

```
// 设备主动获取影子,回调函数会触发onShadow方法，返回设备影子信息
device.getShadow();
```

该函数调用之后将会触发物联网平台将影子发送给设备，在onShadow()中收到的数据格式示例如下：

```
{
"method": "reply",
"payload": {
"status": "success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired": {
"color": {
"timestamp": 1469564492
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

用户可以解析”payload”->”state”中的”reported”得知设备以前上报的影子，分析”desired”得知云端主动修改的影子的数值

云端修改影子数值

物联网平台上的应用程序可能修改影子的数值，在这种情况下云端会将影子数据发送给设备，onShadow()中的res数据格式示例如下：

```
{
"method": "control",
"payload": {
"status": "success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
```

```
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired": {
"color": {
"timestamp": 1469564576
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

用户可以通过“method”的数值为“control”得知这时云端更新了影子。同时通过”state”->”desired”得知修改的内容

删除影子

API原型	iot.device#deleteShadow(keys)
功能说明	删除设备影子
参数说明	- keys：需要删除的设备影子的属性的数组

代码示例：

```
// 删除影子的单个属性
device.deleteShadow("a");
// 删除影子的多个属性
device.deleteShadow(["a","b"]);
// 删除影子的所有属性
device.deleteShadow()
```

示例代码

Github提供了代码示例供开发者参考。