

SmartCloudMobileSDK 接口说明文档

版本：v1.3.2

发布日期：2018-8-15

JD Smart Software doc

版本记录

序号	版本	更改说明	日期	人员
1	v1.0	初稿	2017-03-30	杨长安
2	v1.0	初始化和授权登录	2017-04-21	杨长安
3	v1.0	设备配网、设备控制	2017-05-15	杨长安
4	v1.1	网关子设备、场景、IFTTT	2017-10-10	杨长安
5	v1.2	安全认证	2017-12-20	杨长安
6	v1.3	开放 API	2018-03-12	杨长安
7	V1.3.1	卡片、三级类目	2018-06-15	杨长安
8	V1.3.2	短信验证码授权	2018-08-15	杨长安

目录

1	概述	6
1.1	文档说明	6
2	快速接入流程	6
2.1	获取 AppKey	6
2.2	获取 SmartCloudMobileSDK	6
2.3	快速集成	6
3	初始化和授权登录	7
3.1	时序图	8
3.2	初始化管理器 JDSmartSDK	8
3.2.1	初始化 appKey (必须)	8
3.2.2	设置服务器地址	9
3.2.3	获取 SDK 版本	9
3.3	授权登录管理器	9
3.3.1	AuthorizeCallback	9
3.3.2	京东账号授权	10
3.3.3	短信验证码授权	10
3.3.4	注册授权 token (必须)	11
4	业务类接口	11
4.1.1	获取卡片列表	11
4.1.2	卡片快捷控制	14
4.1.3	三级类目-获取品类列表	15
4.1.4	三级类目-获取产品或品牌列表	17
4.1.5	三级类目-获取产品列表	18
5	设备配网、激活绑定	19
5.1	时序图	20
5.2	设备配网云端接口	20
5.2.1	解析设备二维码信息	20
5.2.2	获取产品信息	21
5.2.3	获取产品配网说明	22
5.2.4	获取设备绑定状态	22
5.2.5	解绑设备	23
5.3	设备配网	23
5.3.1	开始设备发现配置	24
5.3.2	停止设备发现配置	25
5.3.3	开始一键配置	25
5.3.4	停止一键配置	25
5.3.5	开始标准一键配置	26

5.3.6	停止标准一键配置.....	26
5.3.7	开始 SoftAp 配置.....	27
5.3.8	停止 SoftAp 配置.....	27
5.3.9	开始扫描 ble 配网设备.....	27
5.3.10	停止扫描 ble 配网设备.....	28
5.3.11	Ble 单个设备配网.....	28
5.3.12	Ble 批量设备配网.....	29
5.3.13	Ble 批量设备配网.....	30
5.3.14	停止 ble 配置.....	31
5.4	设备激活绑定.....	31
5.4.1	设备激活绑定接口.....	31
5.4.2	设备激活绑定接口.....	32
5.5	设备发现	32
5.5.1	WiFi 设备发现	33
5.5.2	开始设备发现.....	33
5.5.3	停止设备发现.....	33
6	设备控制	34
6.1	设备控制管理器 DeviceControlManager.....	34
6.1.1	获取设备列表.....	34
6.1.2	获取网关设备列表.....	34
6.1.3	修改我的设备名字.....	35
6.1.4	获取设备快照信息.....	35
6.1.5	获取设备详细信息.....	36
6.1.6	获取设备定时数量.....	37
6.1.7	解绑设备.....	37
6.1.8	初始化 js 桥链接.....	38
6.1.9	断开 js 桥链接.....	38
6.2	DeviceControlHandler 接口介绍.....	39
6.2.1	配置导航栏.....	39
6.2.2	打开 url 网页.....	39
6.2.3	跳转添加子设备列表.....	39
6.2.4	网关页面跳转到子设备页面.....	40
6.2.5	配置 ActionBar	40
6.2.6	关闭当前页面.....	41
6.2.7	跳转到本地页面.....	41
6.2.8	分享	41
6.2.9	是否显示标题.....	41
6.2.10	是否显示加载框.....	42
6.2.11	弹框	42
6.2.12	toast 提示.....	42
7	网关子设备	42

7.1	添加子设备云端接口.....	43
7.1.1	获取网关子设备列表.....	43
7.1.2	获取产品配置操作说明.....	43
7.2	添加网关子设备.....	44
7.2.1	解析设备二维码信息.....	44
7.2.2	添加网关子设备：扫码添加.....	44
7.2.3	开始扫描网关子设备.....	45
7.2.4	停止扫描网关子设备.....	46
7.2.5	添加网关子设备：物理按键添加.....	46
8	场景	47
8.1	场景云端接口.....	47
8.1.1	获取支持场景的设备列表.....	47
8.1.2	获取场景列表.....	48
8.1.3	获取场景详情.....	49
8.1.4	创建场景.....	51
8.1.5	删除场景.....	52
8.1.6	更新场景.....	53
8.1.7	执行场景.....	55
8.1.8	停止执行场景.....	55
8.1.9	获取场景执行记录.....	56
8.1.10	获取场景记录详情.....	57
9	IFTTT	59
9.1	IFTTT 云端接口	59
9.1.1	获取支持 IFTTT 的设备列表	59
9.1.2	获取脚本列表.....	60
9.1.3	创建/修改脚本.....	60
9.1.4	停用并删除脚本.....	61
9.1.5	启动脚本.....	62
9.1.6	停止脚本.....	62
9.1.7	检查脚本的状态.....	63
9.1.8	手动执行场景.....	63
9.1.9	获取场景执行日志.....	64
9.1.10	获取用户场景执行记录.....	65
10	开放 API.....	65
10.1	时序图	65
10.2	post 请求.....	66
11	相关错误码定义.....	67

1 概述

1.1 文档说明

本文档面向 Android 开发者。

本文档用于指导开发者快速接入 SmartCloudMobileSDK 提供了 SmartCloudMobileSDK 的各个接口的定义与详细描述，目前 SDK 支持 Android4.0.3 及以上系统，BLE 相关功能需要 Android4.3 及以上系统。

2 快速接入流程

在使用 SDK 之前先注册 appKey (平台注册时生成 appKey 信息) 以及确定客户端用户是否授权成功，授权成功后将 access_token 注册到 SDK。

2.1 获取 AppKey

在京东智能服务平台注册应用获取 appKey，具体操作详见官网获取 appKey 流程说明。

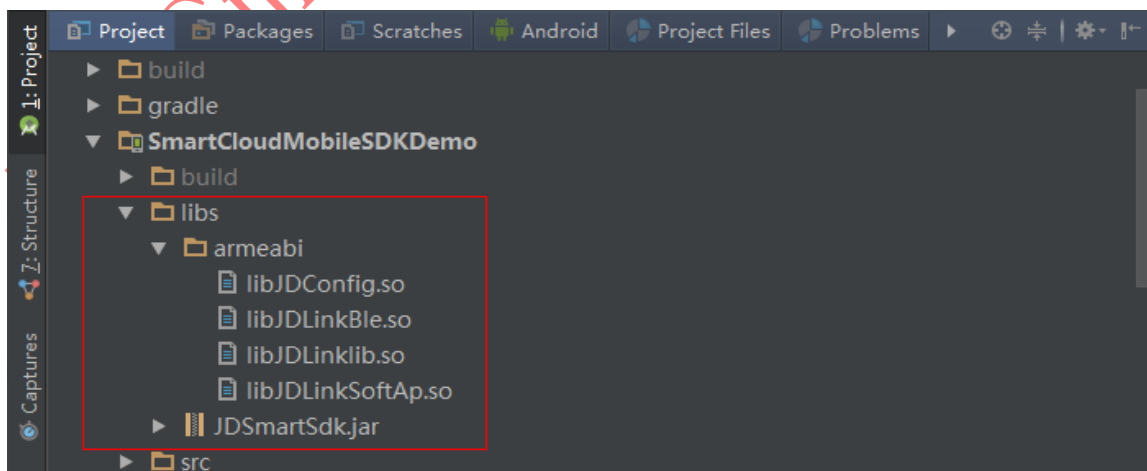
2.2 获取 SmartCloudMobileSDK

在京东智能服务平台下载平台化 SDK。

2.3 快速集成

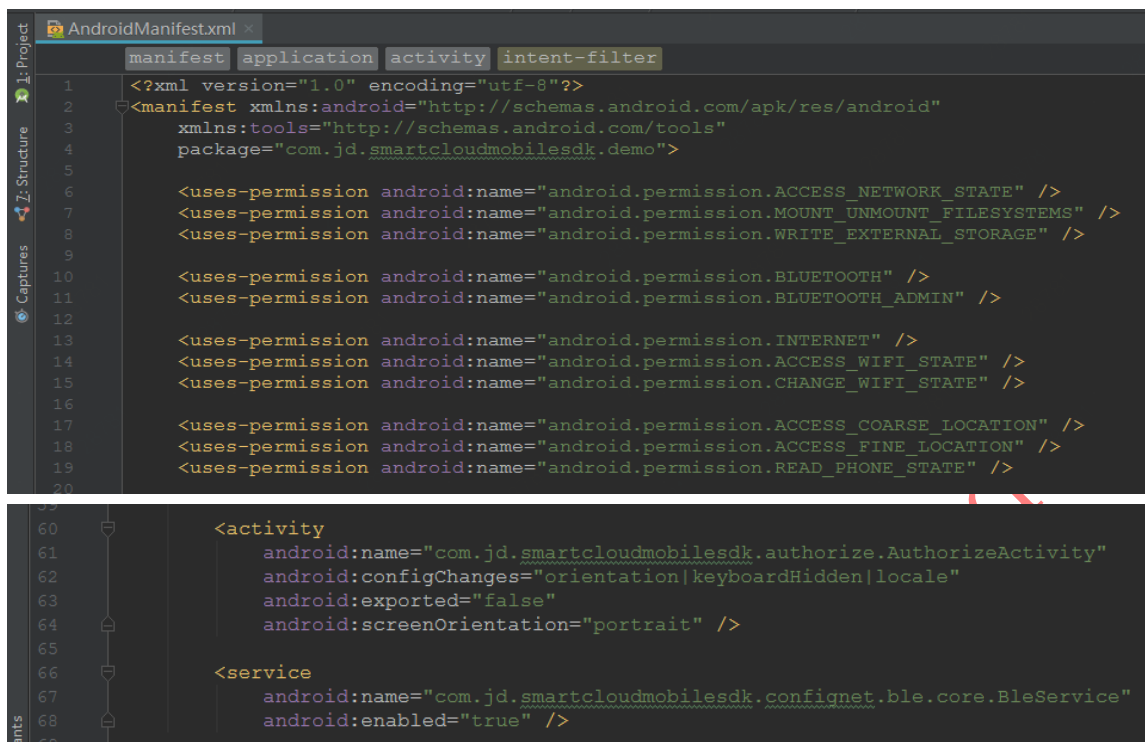
解压 SDK 后首先将 jar 和 so 文件放到项目 libs 文件夹下，并配置相应信息。

步骤一：将 jar 和 so 文件放到项目 libs 文件夹下，如下图：



步骤二：将 jar 添加到项目中，右键点击 jar，在弹出的菜单中点击 Add As Library。

步骤三：配置 AndroidManifest.xml：加入权限和注册 Activity、Service 等，如下图：



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.jd.smartcloudmobilesdk.demo">

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <activity
        android:name="com.jd.smartcloudmobilesdk.authorize.AuthorizeActivity"
        android:configChanges="orientation|keyboardHidden|locale"
        android:exported="false"
        android:screenOrientation="portrait" />

    <service
        android:name="com.jd.smartcloudmobilesdk.confignet.ble.core.BleService"
        android:enabled="true" />
</manifest>
```

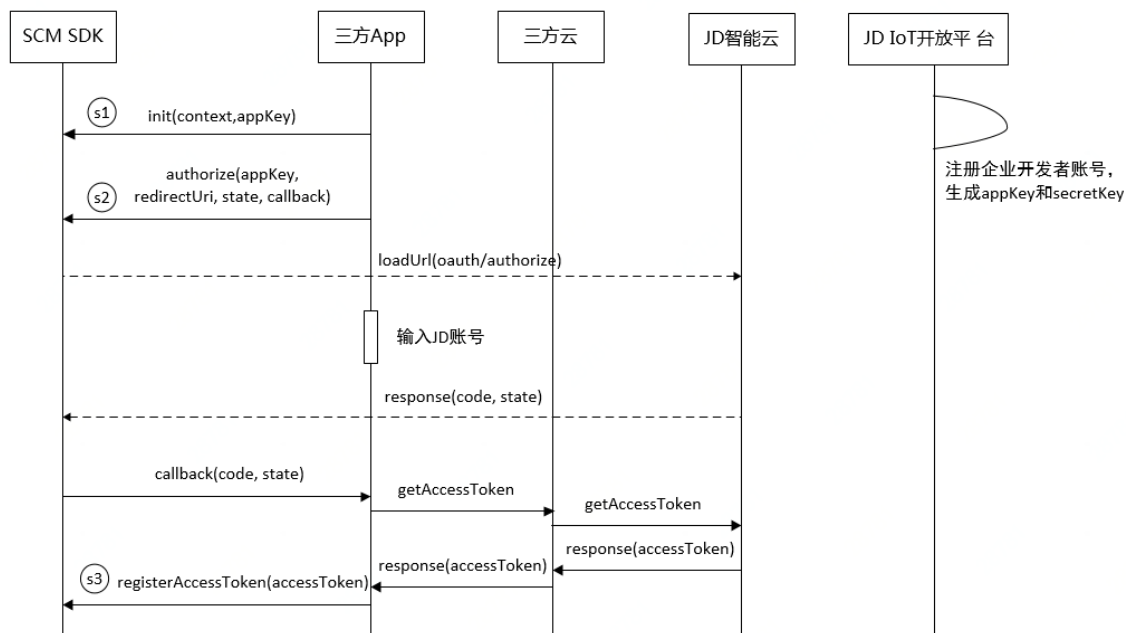
步骤四：集成 SCM Android SDK 的混淆

```
-keep class com.jd.smart.jdlink.** {*;}  
-keep class com.jd.smartcloudmobilesdk.** {*};
```

3 初始化和授权登录

在使用 SDK 之前先注册 appKey(平台注册时生成 appKey 信息) 以及确定客户端用户是否授权成功，授权成功后将 access_token 注册到 SDK。

3.1 时序图



3.2 初始化管理器 JDSmartSDK

JDSmartSDK 用于初始化 Context 和 appKey 等。

详见 Demo 中 JDApplication 中 onCreate 的 init 初始化方法调用。

3.2.1 初始化 appKey (必须)

- 接口签名：
public void init(Context context, String appKey)
- 接口说明：app 启动时，初始化 Context 和 appkey
- 参数说明：

参数名	参数类型	必填	描述
context	Context	是	Application 的上下文对象
appKey	String	是	平台注册时生成 appkey 信息

- 示例代码：
JDSmartSDK.getInstance().init(this, appKey);
初始化 Application 的上下文对象和 appkey，详见 Demo。

3.2.2 设置服务器地址

- 接口签名：

```
public void setServer(int server)
```

- 接口说明：设置服务端环境，在 `init()` 之前调用才能生效，不调用默认是正式环境。

- 参数说明：

参数名	参数类型	必填	描述
server	int	是	默认是正式环境 SERVER_ONLINE 正式环境 SERVER_BOX 沙箱环境

- 示例代码：

```
JDSmartSDK.getInstance().setServer(Constant.SERVER_BOX);
```

3.2.3 获取 SDK 版本

- 接口签名：

```
public String getVersion()
```

- 接口说明：获取 SDK 的版本

- 参数说明：无

- 示例代码：

```
JDSmartSDK.getInstance().getVersion();
```

3.3 授权登录管理器

AuthorizeManager 用于显示用户授权页面以及授权 code 处理等功能 将获取到的 token 注册到 SDK 中，才能使用 SDK 相关接口。本节主要描述授权登录方式获取 token 的过程，开发者也可通过其他授权方式（如二维码授权）获取 token 并注册到 SDK 中。注：**AuthorizeFragment** 提供的授权接口开发者可自定义授权页面样式（标题）。

3.3.1 AuthorizeCallback

AuthorizeCallback 的定义如下：

```
// 获取授权码返回结果接口回调
```

```
public interface AuthorizeCallback {
```

```
    // code: 授权码，state: 自定义，颁发授权后原值返回
```

```
    void onResponse(String code, String state);
```

```
}
```

3.3.2 京东账号授权

➤ 接口签名：

```
public void authorize(String appKey, String redirectUri, String state,
    AuthorizeCallback callback)
```

➤ 接口说明：账号授权，判断 token 不存在时，可调用该接口获取 token

➤ 参数说明：

参数名	参数类型	必填	描述
appKey	String	是	平台注册时生成 appKey 信息
redirectUri	String	是	平台注册时填写的回调地址
state	String	是	自定义，颁发授权后原值返回

➤ 示例代码：

```
AuthorizeManager.getInstance().authorize(appKey, redirectUri, state, new
    AuthorizeCallback(){
        @Override
        public void onResponse(String code, String state) {
        }
    });
```

3.3.3 短信验证码授权

➤ 接口签名：

```
public void smsAuthorize(String appKey, String redirectUri, String state,
    String phoneNumber, String screenMatch, AuthorizeCallback callback)
```

➤ 接口说明：短信验证码授权，判断 token 不存在时，可调用该接口获取 token

➤ 参数说明：

参数名	参数类型	必填	描述
pKey	String	是	平台注册时生成 appkey 信息
redirectUri	String	是	平台注册时填写的回调地址
state	String	是	自定义，颁发授权后返回原值
phoneNumber	String	否	获取验证码和登录的手机号码
screenMatch	String	是	屏幕比例适配参数，由 屏幕比例 和 页面样式 两个数据拼接：R (ratio) 加比例数字表示屏幕比例参数 (eg : R169 表示 16:9 , R83 表示 8:3...) ;

			<p>S (style) 加数字表示页面样式参数 (eg : S0、S1、S2...), 其中 S0 表示默认样式; 参数示例:</p> <p>"R169S1" 表示 : 屏幕比例为 16:9 , 页面样式为 1 , 字母不区分大小写</p>
--	--	--	---

➤ 示例代码 :

```
AuthorizeManager.getInstance().smsAuthorize (appKey, redirectUri, state,
phoneNumber, screenMatch, new AuthorizeCallback(){
    @Override
    public void onResponse(String code, String state) {
    }
});
```

3.3.4 注册授权 token (必须)

➤ 接口签名 :

```
public void registerAccessToken(String accessToken)
```

➤ 接口说明 : 用户授权成功后, 注册授权 token 到 SDK

➤ 参数说明 :

参数名	参数类型	必填	描述
token	String	是	用户授权 token

➤ 示例代码 :

```
AuthorizeManager.getInstance().registerAccessToken(accessToken);
```

使用SDK功能时用户token不存在时, 可调用账号授权或短信验证码授权接口, 授权成功后注册授权token到SDK。

4 业务类接口

BusinessManager 业务类接口管理器, 主要提供了开放服务相关的业务接口, 比如卡片列表、卡片快捷控制、三级类目等业务。

4.1.1 获取卡片列表

➤ 接口签名 :

```
public static void getListCards(ResponseCallback callback)
```

➤ 接口说明 : 获取卡片列表

➤ 参数说明：

请求参数	无
响应结果	<pre>{ "device_total": 4, "scene_total": 1, "cache_time": "yyyy-MM-dd'T'HH:mm:ssZ", "cards": [{ "card_id": 10000, "seq": 1, "card_type": 101, "scene_id": 14400001, "card_name": "回家场景", "c_img_url": ["http://img.360buying.com/xxx.jpg", "http://img.360buying.com/xxx.jpg", "http://img.360buying.com/xxx.jpg"], "card_control": "sence", "status": null, "card_desc": null, "add_card_contrl": null, "add_card_desc": null, "share_from": null, "create_time": "yyyy-MM-dd'T'HH:mm:ssZ" }, { "card_id": 10001, "seq": 2, "card_type": 1, "feed_id": 14440000000001, "card_name": "xx 晾衣架", "c_img_url": ["http://img.360buying.com/xxx.jpg"], }, }</pre>

	<pre> "card_control": null, "status": "1", "card_desc": "设备已连接", "add_card_ctrl": ["down", "pause", "up"], "add_card_desc": null, "share_from": "jd_xxx", "create_time": "yyyy-MM-dd'T'HH:mm:ssZ" }, { "card_id": 10002, "seq": 3, "card_type": 3, "feed_id": 144400000000003, "card_name": "电热水器", "c_img_url": ["http://img.360buying.com/xxx.jpg"], "card_control": "power", "status": "1", "card_desc": "设定温度:60°C", "add_card_ctrl": null, "add_card_desc": ["50°C 实际温度", "50% 剩余水量"], "share_from": "jd_xxx", "create_time": "yyyy-MM-dd'T'HH:mm:ssZ" }, { "card_id": 10003,</pre>
--	--

	<pre> "seq": 4, "card_type": 4, "feed_id": 14440000000004, "card_name": "XX 开关", "c_img_url": ["http://img.360buying.com/xxx.jpg"], "card_control": "power", "status": "1", "card_desc": "已开启", "add_card_contrl": null, "add_card_desc": null, "share_from": "jd_xxx", "create_time": "yyyy-MM-dd'T'HH:mm:ssZ" }] }</pre>
--	---

➤ 示例代码：

```

BusinessManager.getListCards(new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

4.1.2 卡片快捷控制

➤ 接口签名：

```

public static void controlCard(Map<String, Object> params, ResponseCallback
callback) {
```

➤ 接口说明：卡片上的快捷控制，可控制设备也可执行场景

➤ 参数说明：

请求参数	{
------	---

	<pre> "cardType": 1, // 如果为场景则为 101 "cardId": 1000005, "relativeId": "1500000000000005", // 如果为场景则为场景 ID "command": [{ // 控制命令，如果为场景 command 为空 "stream_id": "power", "options": [{ "0": "关" }, { "1": "开" }] }] </pre>
响应结果	<pre> { "status": 0, "error": null, "result": null } </pre>

➤ 示例代码：

```

BusinessManager.controlCard (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
})
@Override
public void onFailure(String response) {
}
});

```

4.1.3 三级类目-获取品类列表

➤ 接口签名：

```

public static void getPopularCategoryList(Map<String, Object> params,
ResponseCallback callback)

```

➤ 接口说明：三级类目-获取品类列表，包含热门品类和全部设备品类。

➤ 参数说明：

请求参数	{ "page": 1, "pageSize": 90 }
响应结果	{ "status": 0, "error": null, "result": { "cate_list": [{ "img_url": "***/57bbbd2N26c63616.png", "cid": 102010, "cname": "插座" }], "pop_cate_list": [{ "img_url": "***/57bbbd7aN64dd3902.png", "cid": 101004, "cname": "洗衣机" }] } }

➤ 示例代码：

```
BusinessManager.getPopularCategoryList(params, new ResponseCallback() {  
    @Override  
    public void onSuccess(String response) {  
        if (CommonUtil.isSuccess(response)) {  
        }  
    }  
    @Override  
    public void onFailure(String response) {  
    }  
});
```


4.1.4 三级类目-获取产品或品牌列表

➤ 接口签名：

```
public static void getProductOrBrand(Map<String, Object> params,  
ResponseCallback callback)
```

➤ 接口说明：根据品类 id 获取产品或品牌信息

➤ 参数说明：

请求参数	<pre>{ "cid": "102003", // 品类 id "page": 1, "pageSize": 60 }</pre>
响应结果	<pre>{ "status": 0, "error": null, "result": { "type": "brand", // "brand"或"product" "brand": [{ "name": "海尔", "brand_id": "1", "cid": "102003" }, { "name": "美的", "brand_id": "2", "cid": "102003" }] }</pre>

➤ 示例代码：

```
BusinessManager.getProductOrBrand(params, new ResponseCallback() {  
    @Override  
    public void onSuccess(String response) {  
        if (CommonUtil.isSuccess(response)) {  
        }  
    }  
})
```

```
    }  
    @Override  
    public void onFailure(String response) {  
    }  
});
```

4.1.5 三级类目-获取产品列表

➤ 接口签名：

```
public static void getProductInfos(Map<String, Object> params, ResponseCallback  
callback)
```

➤ 接口说明：根据品类和品牌 id 获取产品信息

➤ 参数说明：

请求参数	{ "cid": "102003", // 品类 id "brand_id": "2", // 品牌 id "page": 1, "pageSize": 60 }
响应结果	{ "status": 0, "error": null, "result": [{ "id": 26946, "img_url": "****/5603bce4Nb238e140.png", "product_uuid": "U9692E", "config_type": 1107, "name": "美的扫地机器人", "brand_id": 2, "product_models": ["R3-L061C"], "cid": 102003 }] }

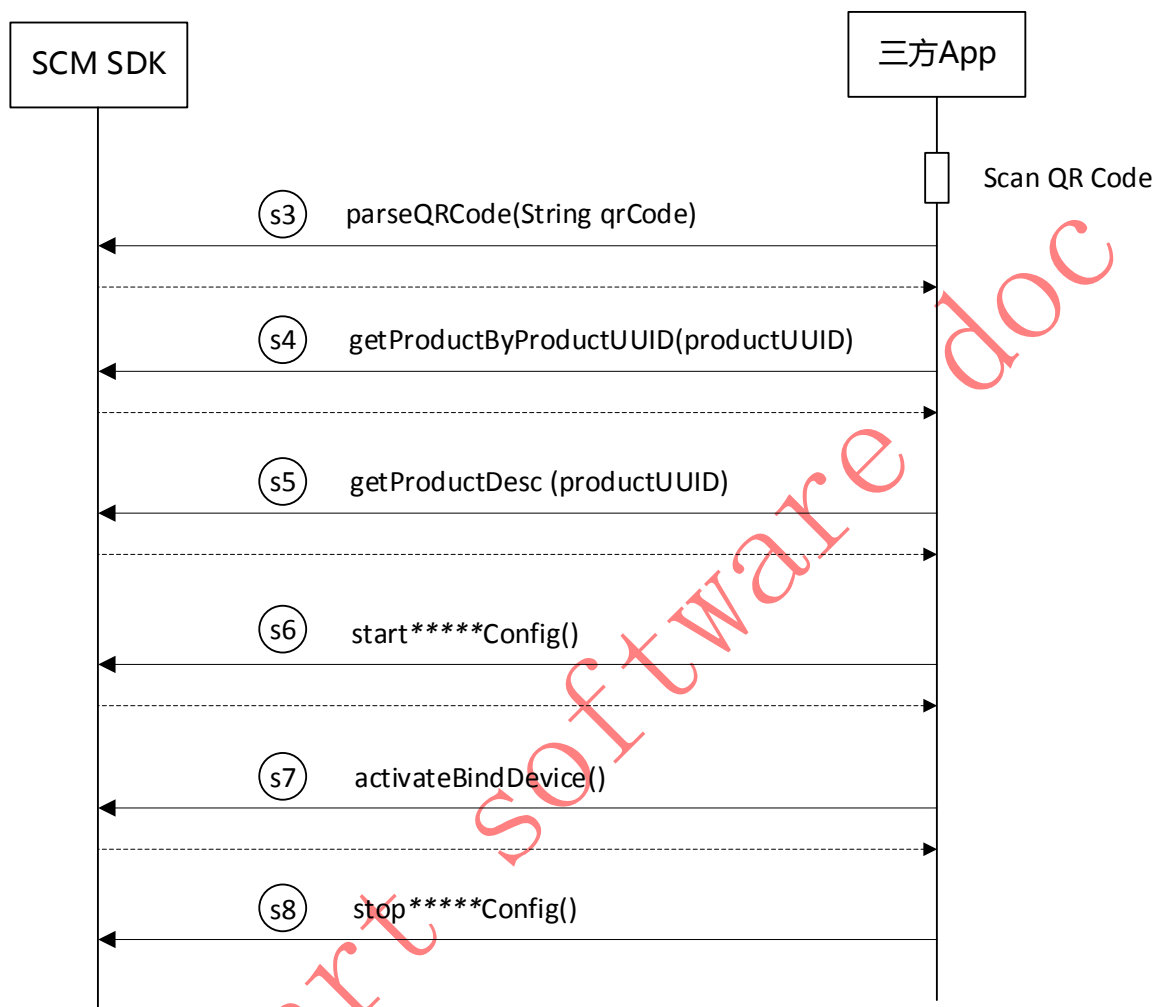
➤ 示例代码：

```
BusinessManager.getProductInfos(params, new ResponseCallback() {  
    @Override  
    public void onSuccess(String response) {  
        if (CommonUtil.isSuccess(response)) {  
            }  
        }  
    @Override  
    public void onFailure(String response) {  
    }  
});
```

5 设备配网、激活绑定

ConfigNetManager 设备配网管理器，主要提供了配网所需的京东云接口和配网 API。京东云接口主要包含获取产品信息、获取产品配网说明、获取设备绑定状态和解绑设备等，配网 API 主要包含一键配置、标准一键配置、SoftAp、Ble+WiFi 配网等。

5.1 时序图



5.2 设备配网云端接口

5.2.1 解析设备二维码信息

➤ 接口签名：

```
public static Map<String, String> parseQRCode(String qrCode)
```

➤ 接口说明：解析设备二维码信息。在设备配网之前，需要扫描设备的二维码（扫码功能自行实现），扫描到的二维码信息通过该接口获取设备相关信息。比如：product_uuid 等

➤ 参数说明：

参数名	参数类型	必填	描述
qrCode	String	是	设备二维码文本信息

➤ 返回值说明：

字段名	字段类型	必填	描述
-----	------	----	----

token	String	否	Token (设备分享)
feed_id	String	否	设备的唯一标识 (设备分享)
device_id	String	否	设备 ID
device_mac	String	否	设备 Mac
device_type	String	否	设备类型
product_uuid	String	否	产品品类标识 puid

- 示例代码：

```
Map<String, String> map = ConfigNetManager.parseQRCode(qrCode);
String product_uuid = map.get(Constant.KEY_PRODUCT_UUID);
```

5.2.2 获取产品信息

- 接口签名：

```
public static void getProductByProductUUID(String productUUID, ResponseCallback
callback)
```

- 接口说明：获取产品信息。在设备配网之前，需要先通过该接口请求必要的产品信息，获取配网方式等关键字段之后方可调用接下来的配网方法。接口入参 puid 可通过 4.2.1 接口获取。

- 参数说明：

参数名	参数类型	必填	描述
productUUID	String	是	产品的 product_uuid(puid)
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
ConfigNetManager.getProductByProductUUID(productUuid, new
ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
}
@Override
public void onFailure(String response) {
}
});
```

5.2.3 获取产品配网说明

➤ 接口签名：

```
public static void getProductDesc (String productUUID, ResponseCallback callback)
```

➤ 接口说明：获取产品配网说明。该接口返回产品配网操作说明，如：设备如何进入配网状态等引导用户完成配网操作，需使用 WebView 展示。接口入参 puid 可通过 4.2.1 接口获取。

➤ 参数说明：

参数名	参数类型	必填	描述
productUUID	String	是	产品的 product_uuid(puid)
callback	ResponseCallback	是	京东云接口回调

➤ 示例代码：

```
ConfigNetManager.getProductDesc(productUUID, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

5.2.4 获取设备绑定状态

➤ 接口签名：

```
public static void getBindStatus(JSONArray array, ResponseCallback callback)
```

➤ 接口说明：获取设备的绑定状态，查询设备和账号的绑定关系。

➤ 参数说明：

参数名	参数类型	必填	描述
array	JSONArray	是	待查询的 feedId 数组
callback	ResponseCallback	是	京东云接口回调

➤ 示例代码：

```
ConfigNetManager.getBindStatus (array, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
```

```

        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
}
});

```

5.2.5 解绑设备

- 接口签名：

```
public static void unbindDevice(String feed_id, int force, ResponseCallback callback)
```

- 接口说明：解绑设备，解除设备和账号的绑定关系。

- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feed_id
force	int	否	是否强制解绑，1：强制
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```

ConfigNetManager.unbindDevice(feed_id, force, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
}
@Override
public void onFailure(String response) {
}
});

```

5.3 设备配网

ConfigNetManager 相关方法用于把支持 Joylink 协议的设备配置到路由器上，建议使用流程：先根据产品 UUID (product_uuid) (二维码获取) 获取产品信息，然后获取产品配网说明引导用户进行设备配网，根据设备配网类型 (config_type) 选择对应的配网方式，SDK 提供多种配网方式，可自由组合使用。详见 Demo 中 ConfigNetActivity 的具体用法。

● 配网类型和 config_type 的对应关系：

配网类型	config_type
一键配置	除下面以外 “11” 开头的，比如 “11**”
标准一键配置	1113
SoftAp 配网	1114
SoftAp + 标准一键配置	1115
BLE 配网	1116
BLE + 标准一键配置	1117
ThunderConfig + 标准一键配置	1118
ThunderConfig + SoftAp	1119
自助入网设备：如电视类	1903

5.3.1 开始设备发现配置

➤ 接口签名：

```
public void startScanConfig(String productUUID, WiFiConfigCallback callback)
```

➤ 接口说明：开始设备发现配置。 该方式适合已入网或自助入网设备（如电视类），接口内部不做设备配网，只进行设备发现，设备发现的结果通过回调返回。为了统一多种配网方式，方便调用，该接口对 4.5 设备发现进行封装，适合已入网设备使用。

➤ 参数说明：

参数名	参数类型	必填	描述
productUUID	String	是	产品 UUID
callback	WiFiConfigCallback	是	配网成功的设备列表回调

➤ 示例代码：

```
ConfigNetManager.getInstance().startScanConfig(productUUID, new
WiFiConfigCallback() {
    @Override
    public void onScanResult(List<WiFiScanDevice> deviceList) {
    }

    @Override
    public void onConfigFailed(String error) {
    }
});
```


5.3.2 停止设备发现配置

- 接口签名：

```
public void stopScanConfig()
```

- 接口说明：停止设备发现配置

- 参数说明：无

- 示例代码：

```
ConfigNetManager.getInstance().stopScanConfig ();
```

5.3.3 开始一键配置

- 接口签名：

```
public void startOneStepConfigCloud(OneStepCloudModel model,  
WiFiConfigCallback callback)
```

- 接口说明：开始一键配置。**注：**产品二维码中包含 deviceMac 的，一定要设置到配网参数 model 中，比如霍尼韦尔设备。

- 参数说明：

参数名	参数类型	必填	描述
model	OneStepCloudModel	是	配网所需的数据
callback	WiFiConfigCallback	是	配网成功的设备列表回调

- 示例代码：

```
OneStepCloudModel model = new OneStepCloudModel(wifiSSID, wifiPwd,  
productUUID, configType, deviceMac);  
ConfigNetManager.getInstance().startOneStepConfigCloud(model, new  
WiFiConfigCallback() {  
    @Override  
    public void onScanResult(List<WiFiScanDevice> deviceList) {  
    }  
    @Override  
    public void onConfigFailed(String error) {  
    }  
});
```

5.3.4 停止一键配置

- 接口签名：

```
public void stopOneStepConfigCloud()
```

- 接口说明：停止一键配置
- 参数说明：无
- 示例代码：

```
ConfigNetManager.getInstance().stopOneStepConfigCloud();
```

5.3.5 开始标准一键配置

- 接口签名：

```
public void startOneStepConfigNative (OneStepNativeModel model,
    WiFiConfigCallback callback)
```

- 接口说明：开始标准一键配置
- 参数说明：

参数名	参数类型	必填	描述
model	OneStepNativeModel	是	配网所需的数据
callback	WiFiConfigCallback	是	配网成功的设备列表回调

- 示例代码：

```
OneStepNativeModel model = new OneStepNativeModel(wifiSSID, wifiPwd,
    productUUID);
ConfigNetManager.getInstance().startOneStepConfigNative (model, new
    WiFiConfigCallback() {
        @Override
        public void onScanResult(List<WiFiScanDevice> deviceList) {
        }
        @Override
        public void onConfigFailed(String error) {
        }
    });
```

5.3.6 停止标准一键配置

- 接口签名：

```
private void stopOneStepConfigNative ()
```

- 接口说明：停止标准一键配置
- 参数说明：无
- 示例代码：

ConfigNetManager.getInstance().stopOneStepConfigNative ();

5.3.7 开始 SoftAp 配置

- 接口签名：

```
public void startSoftApConfig (SoftApModel model, WiFiConfigCallback callback)
```

- 接口说明：开始 SoftAp 配置

- 参数说明：

参数名	参数类型	必填	描述
model	SoftApModel	是	配网所需的数据
callback	WiFiConfigCallback	是	配网成功的设备列表回调

- 示例代码：

```
SoftApModel model = new SoftApModel (wifiSSID, wifiPwd, productUUID);
ConfigNetManager.getInstance().startSoftApConfig (model, new
WiFiConfigCallback() {
    @Override
    public void onScanResult(List<WiFiScanDevice> deviceList) {
    }
    @Override
    public void onConfigFailed(String error) {
    }
});
```

5.3.8 停止 SoftAp 配置

- 接口签名：

```
private void stopSoftApConfig ()
```

- 接口说明：停止 SoftAp 配置

- 参数说明：无

- 示例代码：

```
ConfigNetManager.getInstance().stopSoftApConfig ();
```

5.3.9 开始扫描 ble 配网设备

- 接口签名：

```
public void bleStartScan(BleScanCallback callback)
```

- 接口说明：扫描支持 ble 配网的设备（在 ble 配网之前调用）

- 参数说明：

参数名	参数类型	必填	描述
callback	BleScanCallback	是	Ble 扫描设备接口回调

- 示例代码：

```
ConfigNetManager.getInstance().bleStartScan(mBleScanCallback);
private BleScanCallback mBleScanCallback = new BleScanCallback() {
    @Override
    public void onScanResult(final BleDevice bleDevice) {
    }
};
```

5.3.10 停止扫描 ble 配网设备

- 接口签名：

```
public void bleStopScan()
```

- 接口说明：停止扫描 ble 配网设备

- 参数说明：无

- 示例代码：

```
ConfigNetManager.getInstance().bleStopScan();
```

5.3.11 Ble 单个设备配网

- 接口签名：

```
public void bleSingleConfig(String wifiSSID, String wifiPwd, BleDevice bleDevice,
    BleConfigCallback callback)
```

- 接口说明：开始 Ble 单个设备配网

- 参数说明：

参数名	参数类型	必填	描述
wifiSSID	String	是	WiFi 的 SSID
wifiPwd	String	是	WiFi 的密码
bleDevice	BleDevice	是	待配网的 Ble 设备
callback	BleConfigCallback	是	配网接口回调

- 示例代码：

```
ConfigNetManager.getInstance().bleSingleConfig(wifiSSID, wifiPwd, bleDevice,
    mBleConfigCallback);
```

```

private BleConfigCallback mBleConfigCallback = new BleConfigCallback() {
    @Override
    public void onWiFiStatus(String address, final int wifiStatus) {
    }
    @Override
    public void onDebugLog(String log) {
    }
    @Override
    public void onTimerTick(long millisInFuture, long millisUntilFinished) {
    }
    @Override
    public void onScanResult(final List< WiFiScanDevice > deviceList) {
    }
};

```

5.3.12 Ble 批量设备配网

➤ 接口签名：

```

public void bleMultiConfig(String wifiSSID, String wifiPwd, BleDevice bleDevice,
    BleConfigCallback callback)

```

➤ 接口说明：Ble 批量设备配网。

➤ 参数说明：

参数名	参数类型	必填	描述
wifiSSID	String	是	WiFi 的 SSID
wifiPwd	String	是	WiFi 的密码
bleDevice	BleDevice	是	待配网的 Ble 设备
callback	BleConfigCallback	是	配网接口回调

➤ 示例代码：

```

ConfigNetManager.getInstance().bleMultiConfig(wifiSSID, wifiPwd, bleDevice,
    mBleConfigCallback);

```

```

private BleConfigCallback mBleConfigCallback = new BleConfigCallback() {
    @Override
    public void onWiFiStatus(String address, final int wifiStatus) {
    }
    @Override

```

```

    public void onDebugLog(String log) {
    }
    @Override
    public void onTimerTick(long millisInFuture, long millisUntilFinished) {
    }
    @Override
    public void onScanResult(final List< WiFiScanDevice > deviceList) {
    }
};

```

5.3.13 Ble 批量设备配网

➤ 接口签名：

```
public void bleMultiConfig(String wifiSSID, String wifiPwd, List<BleDevice> bleList,
    BleConfigCallback callback)
```

➤ 接口说明：Ble 批量设备配网。

➤ 参数说明：

参数名	参数类型	必填	描述
wifiSSID	String	是	WiFi 的 SSID
wifiPwd	String	是	WiFi 的密码
bleList	List<BleDevice>	是	待配网的 Ble 设备列表
callback	BleConfigCallback	是	配网接口回调

➤ 示例代码：

```

ConfigNetManager.getInstance().bleMultiConfig(wifiSSID, wifiPwd, bleList,
    mBleConfigCallback);
private BleConfigCallback mBleConfigCallback = new BleConfigCallback() {
    @Override
    public void onWiFiStatus(String address, final int wifiStatus) {
    }
    @Override
    public void onDebugLog(String log) {
    }
    @Override
    public void onTimerTick(long millisInFuture, long millisUntilFinished) {
    }
}

```

```

@Override
public void onScanResult(final List< WiFiScanDevice > deviceList) {
}
};

```

5.3.14 停止 ble 配置

- 接口签名：

```
public void stopBleConfig()
```

- 接口说明：停止 ble 配网

- 参数说明：无

- 示例代码：

```
ConfigNetManager.getInstance().stopBleConfig();
```

5.4 设备激活绑定

ActivateManager 把入网成功的设备注册到京东云，绑定到相关账号下。

详见 Demo 中具体用法。

5.4.1 设备激活绑定接口

- 接口签名：

```
public void activateBindDevice(WiFiScanDevice device, BindCallback callback)
```

- 接口说明：设备激活绑定

- 参数说明：

参数名	参数类型	必填	描述
device	WiFiScanDevice	是	配网成功回调的设备数据
callback	BindCallback	是	设备绑定结果接口回调

- 示例代码：

```

ActivateManager.getInstance().activateBindDevice(device, mBindCallback);
private BindCallback mBindCallback = new BindCallback() {
    @Override
    public void onSuccess(WiFiScanDevice device, BindResult bindResult) {
    }
    @Override
    public void onError(WiFiScanDevice device, String error) {

```

```

    }
    @Override
    public void onFailure(String response) {
    }
};

```

5.4.2 设备激活绑定接口

- 接口签名：

```
public void activateBindDevice(List< WiFiScanDevice > deviceList, BindCallback
callback)
```

- 接口说明：设备激活绑定

- 参数说明：

参数名	参数类型	必填	描述
deviceList	List< WiFiScanDevice >	是	配网成功回调的设备数据
callback	BindCallback	是	设备绑定结果接口回调

- 示例代码：

```

ActivateManager.getInstance().activateBindDevice(deviceList, mBindCallback);
private BindCallback mBindCallback = new BindCallback() {
    @Override
    public void onSuccess(WiFiScanDevice device, BindResult bindResult) {
    }
    @Override
    public void onError(WiFiScanDevice device, String error) {
    }
    @Override
    public void onFailure(String response) {
    }
};

```

5.5 设备发现

WiFiScanManager 设备发现管理类，用于发现已入网的设备（集成 JoyLink 协议），设备发现的结果通过返回值或 callback 返回。

5.5.1 WiFi 设备发现

- 接口签名：

```
public List<WiFiScanDevice> scanDevice(String productUUID)
```

- 接口说明：发现已入网的设备，该接口调用一次只进行一次设备发现，大概 5 秒返回结果，未发现符合条件的设备则返回空。如需多次进行设备发现，建议每隔 5 秒调用一次该方法。接口为阻塞性方法，**建议在子线程中调用。**

- 参数说明：

参数名	参数类型	必填	描述
productUUID	String	否	过滤发现的设备，为空则不过滤

- 示例代码：

```
List<WiFiScanDevice> scanList =
    WiFiScanManager.getInstance().scanDevice(productUUID);
```

5.5.2 开始设备发现

- 接口签名：

```
public void startScan(String productUUID, WiFiScanCallback callback)
```

- 接口说明：开始设备发现，发现已入网的设备，结果通过 callback 返回，没有发现设备则不回调，该方法会循环进行设备发现，建议默认超时时间为 60 秒，可以自定义设备发现超时时间，也可以随时停止。

- 参数说明：

参数名	参数类型	必填	描述
productUUID	String	否	过滤发现的设备，为空则不过滤
callback	WiFiScanCallback	是	设备发现回调已入网的设备列表

- 示例代码：

```
WiFiScanManager.getInstance().startScan(productUUID, new WiFiScanCallback() {
    @Override
    public void onScanResult(List<WiFiScanDevice> deviceList) {
    }
});
```

5.5.3 停止设备发现

- 接口签名：

```
public void stopScan()
```

- 接口说明：停止设备发现
- 参数说明：无
- 示例代码：
WiFiScanManager.getInstance().stopScan();

6 设备控制

6.1 设备控制管理器 DeviceControlManager

6.1.1 获取设备列表

- 接口签名：
public static void getDeviceList(ResponseCallback responseCallback)
- 接口说明：获取用户设备列表
- 参数说明：

参数名	参数类型	必填	描述
responseCallback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
DeviceControlManager.getDeviceList(new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

6.1.2 获取网关设备列表

- 接口签名：
public static void getSubDevices(String feed_id, ResponseCallback callback)
- 接口说明：获取网关设备列表
- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feedId，非空
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
DeviceControlManager.getSubDevices(feed_id,new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

6.1.3 修改我的设备名字

- 接口签名：

```
public static void editDeviceName(String name, String feed_id, ResponseCallback
callback)
```

- 接口说明：获取网关设备列表

- 参数说明：

参数名	参数类型	必填	描述
name	String	是	设备新名称，非空
feed_id	String	是	设备的 feedId，非空
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
DeviceControlManager.editDeviceName (name, feed_id, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

6.1.4 获取设备快照信息

- 接口签名：

```
public static void getSnapshotWithFeedId(String feed_id, ResponseCallback
callback)
```

- 接口说明：获取设备快照信息

- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feedId , 非空
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
DeviceControlManager. getSnapshotWithFeedId (feed_id, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

6.1.5 获取设备详细信息

- 接口签名：

```
public void getDeviceInfo(String feed_id, OnDeviceDataLoadListener listener)
```

- 接口说明：获取设备详细信息

- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feedId ,非空
listener	OnDeviceDataLoadListener	是	京东云接口回调

- 示例代码：

```
deviceControlManager.getDeviceInfo(feed_id, new
DeviceControlManager.OnDeviceDataLoadListener() {
    @Override
    public void onDetailLoad(Result result, String response) {
    }
    @Override
    public void onStart() {
    }
    @Override
    public void onFinish() {
```

```

    }
    });

```

6.1.6 获取设备定时数量

- 接口签名：

```
public void getDeviceTimeTaskWithFeedId (String feed_id, ResponseCallback
callback)
```

接口说明：获取设备定时数量

- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feedId，非空
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```

DeviceControlManager. getDeviceTimeTaskWithFeedId (feed_id, new
ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});

```

6.1.7 解绑设备

- 接口签名：

```
public static void unbindDevice(String feed_id, String force, ResponseCallback
callback)
```

- 接口说明：获取设备定时数量

- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	设备的 feedId，非空
force	String	是	是否强行解绑，默认非强制 1：强制
callback	ResponseCallback	是	京东云接口回调

➤ 示例代码：

```
DeviceControlManager.unbindDevice(String feed_id, String force, new
ResponseCallback() {
    @Override
    public void onSuccess(String response) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

6.1.8 初始化 js 桥链接

➤ 接口签名：

```
public void initH5Data(WebView webView, OnDeviceControlListener listener)
```

➤ 接口说明：初始化 js 桥连接，在获取设备信息成功后调用此方法

➤ 参数说明：

参数名	参数类型	必填	描述
webView	WebView	是	展示设备详情的 WebView 对象
listener	OnDeviceControlListener	是	Js 桥到页面的回调

➤ 示例代码：

```
deviceControlManager.initH5Data(webView, new DeviceControlHandler());
```

详见 Demo 中具体用法。

6.1.9 断开 js 桥链接

➤ 接口签名：

```
public void destroy()
```

➤ 接口说明：断开 js 桥链接，在 onDestroy 中调用此方法

➤ 参数说明：无

➤ 示例代码：

```
@Override
protected void onDestroy() {
    if (deviceControlManager != null) {
        deviceControlManager.destroy();
    }
}
```

```
    }  
    super.onDestroy();  
}
```

6.2 DeviceControlHandler 接口介绍

6.2.1 配置导航栏

- 接口签名：

```
public void config(String data)
```

- 接口说明：配置导航栏按钮的显示与隐藏、title 的文字
- 参数说明：data：js 桥传过来的消息 json 字符串，详细见下表

参数名	参数类型	必填	描述
showBack	boolean	否	是否显示后退按钮
showMore	boolean	否	是否显示更多\设置按钮
showOnline	boolean	否	用来控制设备在线状态，true：不在线,false：在线
titletext	String	是	配置页面标题，默认为设备名

6.2.2 打开 url 网页

- 接口签名：

```
public void openUrl(String url)
```

- 接口说明：打开 url 网页
- 参数说明：

参数名	参数类型	必填	描述
url	String	是	网页链接 url

6.2.3 跳转添加子设备列表

- 接口签名：

```
public void addSubDevice ()
```

- 接口说明：跳转到添加子设备列表
- 参数说明：无

6.2.4 网关页面跳转到子设备页面

- 接口签名：

```
public void onRefresh(String p_feed_id, String feed_id)
```

- 接口说明：网关页面跳转到子设备页面

- 参数说明：

参数名	参数类型	必填	描述
p_feed_id	String	是	网关设备的唯一标识
feed_id	String	是	子设备的唯一标识

6.2.5 配置 ActionBar

- 接口签名：

```
public void configActionBar(String data)
```

- 接口说明：配置标题栏中各个按钮的点击事件

- 参数说明：data：js 桥传过来的消息 json 字符串，详细见下表

参数名	参数类型	必填	描述
what	JSONArray	是	表示控件 id，左右两边各 2 个，固定值，依次为 button1-button4，数组为空则显示默认控件
display	JSONArray	是	显示到控件上的文本或图片。和 what 数组的顺序保持一致。以 drawable_ 开始表示为图片。暂支持以下图片 "drawable_back"; "drawable_setting"; "drawable_more"; "drawable_add"; "drawable_close";
callBackName	JSONArray	是	控件的点击事件 js 方法名。和 what 数组的顺序保持一致。当方法名为"goBack"、"close"、"setting"其中一个时将执行 native 方法，不会回调到 js。

6.2.6 关闭当前页面

- 接口签名：
public void finish ()
- 接口说明：关闭当前页面
- 参数说明：无

6.2.7 跳转到本地页面

- 接口签名：
public void jumpNativePage(String data)
- 接口说明：跳转到 App 的本地页面
- 参数说明：

参数名	参数类型	必填	描述
data	String	是	本地页面标识

6.2.8 分享

- 接口签名：
public void shareSTH(JSONObject content)
- 接口说明：App 分享
- 参数说明：content：js 桥传过来的消息 json 字符串，详细见下表

参数名	参数类型	必填	描述
shareUrl	String	是	分享信息的链接
imgUrl	String	是	分享信息的图片链接
title	String	是	分享信息的标题
message	String	是	分享信息的内容

6.2.9 是否显示标题

- 接口签名：
public void showTitle(boolean show)
- 接口说明：是否显示 App 标题 title
- 参数说明：

参数名	参数类型	必填	描述
show	boolean	是	true:显示，false：不显示

6.2.10 是否显示加载框

- 接口签名：
public void showLoading (boolean show)
- 接口说明：是否显示 App 的加载框
- 参数说明：

参数名	参数类型	必填	描述
show	boolean	是	true:显示，false：不显示

6.2.11 弹框

- 接口签名：
public void alert(String data, WVJBResponseCallback jsCallback)
- 接口说明：App 弹框
- 参数说明：data：js 桥传过来的消息 json 字符串，详细见下表

参数名	参数类型	必填	描述
messageTitle	String	是	标题信息
messageYes	String	是	确认按钮信息
messageNo	String	是	取消按钮信息

6.2.12 toast 提示

- 接口签名：
public void toast(String message)
- 接口说明：toast 消息
- 参数说明：

参数名	参数类型	必填	描述
message	String	是	toast 显示的消息

7 网关子设备

GatewayManager 网关子设备管理器，主要提供了添加子设备所需的京东云接口和配网 API。京东云接口主要包含获取网关子设备列表和获取产品配置操作说明。添加子设备 API 支持扫码和物理按键等方式添加。

7.1 添加子设备云端接口

7.1.1 获取网关子设备列表

- 接口签名：

```
public static void getSubDevices(String feed_id, ResponseCallback callback)
```

- 接口说明：获取网关子设备列表，即当前网关下支持的子设备列表
- 参数说明：

参数名	参数类型	必填	描述
feed_id	String	是	网关的 feed_id
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
GatewayManager.getSubDevices (feed_id, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

7.1.2 获取产品配置操作说明

- 接口签名：

```
public static void getProductManual(String product_id, ResponseCallback callback)
```

- 接口说明：获取产品配置操作说明
- 参数说明：

参数名	参数类型	必填	描述
product_id	String	是	网关子设备的 product_id
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
GatewayManager.getProductManual (product_id, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
```

```
        if (CommonUtil.isSuccess(response)) {  
            }  
        }  
        @Override  
        public void onFailure(String response) {  
        }  
    }  
};
```

7.2 添加网关子设备

7.2.1 解析设备二维码信息

- 接口签名：

```
public static Map<String, String> parseQRCode(String qrCode)
```

- 接口说明：解析设备二维码信息。在设备配网之前，需要扫描设备的二维码（扫码功能自行实现），扫描到的二维码信息通过该接口获取设备相关信息。比如：product_uuid 等

- 参数说明：

参数名	参数类型	必填	描述
qrCode	String	是	设备二维码文本信息

- 返回值说明：

字段名	字段类型	必填	描述
token	String	否	Token（设备分享）
feed_id	String	否	设备的唯一标识（设备分享）
device_id	String	否	设备 ID
device_mac	String	否	设备 Mac
device_type	String	否	设备类型
product_uuid	String	否	产品品类标识 puid

- 示例代码：

```
Map<String, String> qrCodes = GatewayManager.parseQRCode(qrCode);  
String product_uuid = qrCodes.get(Constant.KEY_PRODUCT_UUID);  
String device_mac = qrCodes.get(Constant.KEY_DEVICE_MAC);
```

7.2.2 添加网关子设备：扫码添加

- 接口签名：

```
public void addSubDevice(GatewayDevice gatewayDevice, Map<String, String>
```

qrCodes, GatewayBindCallback callback)

- 接口说明：添加网关子设备，适用于扫码添加方式
- 参数说明：

参数名	参数类型	必填	描述
gatewayDevice	GatewayDevice	是	网关设备的数据模型
qrCodes	Map<String, String>	是	网关子设备二维码解析结果
callback	GatewayBindCallback	是	网关子设备绑定结果接口

- 示例代码：

```
GatewayManager.getInstance().addSubDevice(gatewayDevice, qrCodes, new
GatewayBindCallback() {
    @Override
    public void onSuccess(GatewayScanDevice device) {
    }
    @Override
    public void onError(GatewayScanDevice device, GatewayBindError bindError) {
    }
    @Override
    public void onFailure(String response) {
    }
});
```

7.2.3 开始扫描网关子设备

- 接口签名：


```
public void startScan(GatewayDevice gatewayDevice, GatewayScanCallback
callback)
```
- 接口说明：开始扫描网关子设备，即设备发现子设备
- 参数说明：

参数名	参数类型	必填	描述
gatewayDevice	GatewayDevice	是	网关设备的数据模型
callback	GatewayScanCallback	是	扫描网关子设备接口回调

- 示例代码：

```
GatewayManager.getInstance().startScan(gatewayDevice, new
GatewayScanCallback() {
    @Override
```

```

        public void onTimerTick(long millisInFuture, long millisUntilFinished) {
        }
        @Override
        public void onScanResult(List<GatewayScanDevice> deviceList) {
        }
    };

```

7.2.4 停止扫描网关子设备

- 接口签名：

```
public void stopScan()
```

- 接口说明：停止扫描网关子设备
- 参数说明：无
- 示例代码：

```
GatewayManager.getInstance().stopScan();
```

7.2.5 添加网关子设备：物理按键添加

- 接口签名：

```
public void addSubDevice(GatewayDevice gatewayDevice, GatewayScanDevice
subDevice, GatewayBindCallback callback)
```

- 接口说明：添加网关子设备，适用于物理按键添加方式
- 参数说明：

参数名	参数类型	必填	描述
gatewayDevice	GatewayDevice	是	网关设备的数据模型
subDevice	GatewayScanDevice	是	设备发现子设备的数据模型
callback	GatewayBindCallback	是	网关子设备绑定结果接口

- 示例代码：

```

GatewayManager.getInstance().addSubDevice(gatewayDevice, subDevice, new
GatewayBindCallback() {
    @Override
    public void onSuccess(GatewayScanDevice device) {
    }
    @Override
    public void onError(GatewayScanDevice device, GatewayBindError bindError) {
    }
}

```

```
@Override
public void onFailure(String response) {
}

});
```

8 场景

SceneManager 场景管理器，主要提供了场景相关的京东云接口。详见 demo

8.1 场景云端接口

8.1.1 获取支持场景的设备列表

➤ 接口签名：

```
public static void getIFTTTDeviceList(Map<String, Object> params,
ResponseCallback callback)
```

➤ 接口说明：获取支持场景的设备列表

➤ 参数说明：

请求参数	{ "part": "response" // 获取 ifttt 相应设备的参数和设备信息 }
响应结果	{ "status": 0, "error": null, "result": { "list": [{ "p_description": "古北智能插座 SPmini", "device_name": "古北智能插座（新）", "p_img_url": "", "product_id": 20435, "pro_type": 102010, "type": 1, "feed_id": 144309283668308867, "version": "2.0", "stream": [{ "value_type": "string",

	<pre> "symbol": "", "stream_name": "设置开关", "manu_set": [], "value_des": [{ "0": "关" }], { "1": "开" }], "stream_id": "pwr", "max_value": "0.000", "ifttt_value_desc": null, "units": "", "min_value": "0.000" } } } </pre>
--	---

➤ 示例代码：

```

SceneManager.getIFTTTDeviceList (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {

        }
    }
    @Override
    public void onFailure(String response) {

    }
});

```

8.1.2 获取场景列表

➤ 接口签名：

```

public static void getSceneList (Map<String, Object> params, ResponseCallback
callback)

```


- 接口说明：获取场景列表

- 参数说明：

请求参数	
响应结果	<pre>{ "status": 0, "error": null, "result": [{ "id": 123, "name": "场景名字", "status": 1, //1 完成, 2 执行中, 3 取消, 4 服务端丢失该场景, 5 待执行 "start_time": 1525418424000, "end_time": 1525479565000, "images": ["***/56581666Nf44980fd.png"] }] }</pre>

- 示例代码：

```
SceneManager.getSceneList (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

8.1.3 获取场景详情

- 接口签名：

```
public static void getSceneDetail (Map<String, Object> params, ResponseCallback
callback)
```

- 接口说明：获取场景详情

- 参数说明：

请求参数	{ "id ": 123 }
响应结果	<pre> { "status": 0, "error": null, "result": { "status": 1, "items": [{ "id": 1, // 子任务 id "status": -1, "device_type": "time", "delay": 61140 // 延迟时间 (秒), -1 表示立即执行 }, { "id": 2, "device_name": "设备名称", "status": -1, // -1 初始状态, 1 完成, 2 待执行, 3 设备离线, 4 设备已删除, 5 网络异常, 6 执行失败, 7 用户主动取消设备执行, 8 服务端丢失该 任务执行 "device_delete": 4, // 设备已删除, 其他状态客户端暂不做处理 "device_type": "device", "images": "***/58aa8f35N632c159b.jpg", "streams": [{ "current_value": "0", "current_value_zh": "关", "stream_name_zh": "开关", "stream_id": "power", "real_value_zh": "关" }], "feed_id": 149796304011705303 }] } </pre>

➤ 示例代码：

```
SceneManager.getSceneDetail (params, new ResponseCallback() {
```

```

@Override
public void onSuccess(String response) {
    if (CommonUtil.isSuccess(response)) {
    }
}

@Override
public void onFailure(String response) {
}

});

```

8.1.4 创建场景

➤ 接口签名：

```
public static void createScene (Map<String, Object> params, ResponseCallback
callback)
```

➤ 接口说明：创建场景

➤ 参数说明：

请求参数	<pre> { "name": "场景名称", "items": [{ "feed_id": "144309283668308867", "device_type": "device", "streams": [{ "stream_id": "pwr", "current_value": "1" }] }, { "delay": 5, "device_type": "time" }, { "feed_id": "149796304011705303", "device_type": "device", "streams": [{ "stream_id": "power", "current_value": "1" }] } } </pre>
------	--

	<pre> }} }} }</pre>
响应结果	<pre> { "status": 0, "error": null, "result": { "id": 162257 } }</pre>

➤ 示例代码：

```

SceneManager.createScene (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

8.1.5 删除场景

➤ 接口签名：

```
public static void deleteScene (String paramJson, ResponseCallback callback)
```

➤ 接口说明：删除场景

➤ 参数说明：

请求参数	<pre> [{"id": "162253" // 场景 id }</pre>
------	---

响应结果	<pre>{ "status": 0, "error": null, "result": null }</pre>
------	---

➤ 示例代码：

```
SceneManager.deleteScene (paramJson, new ResponseCallback(){
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

8.1.6 更新场景

➤ 接口签名：

```
public static void updateScene (Map<String, Object> params, ResponseCallback
callback)
```

➤ 接口说明：更新场景

➤ 参数说明：

请求参数	<pre>{ "id": "162253", "name": "场景名称", "items": [{ "feed_id": "144309283668308867", "id": "162254", "device_type": "device", "streams": [{ "stream_name": null, "stream_id": "pwr", </pre>
------	--

	<pre> "current_value": "1" } }, { "delay": 5, "id": "162255", "device_type": "time" }, { "feed_id": "149796304011705303", "id": "162256", "device_type": "device", "streams": [{ "stream_name": null, "stream_id": "power", "current_value": "1" }] }] }</pre>
响应结果	<pre> { "status": 0, "error": null, "result": null }</pre>

➤ 示例代码：

```

SceneManager.updateScene (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

8.1.7 执行场景

- 接口签名：

```
public static void executeScene(Map<String, Object> params, ResponseCallback  
callback)
```

- 接口说明：执行场景

- 参数说明：

请求参数	{"id":"162257"}
响应结果	{ "status": 0, "error": null, "result": null }

- 示例代码：

```
SceneManager.executeScene (params, new ResponseCallback() {  
    @Override  
    public void onSuccess(String response) {  
        if (CommonUtil.isSuccess(response)) {  
        }  
    }  
    @Override  
    public void onFailure(String response) {  
    }  
});
```

8.1.8 停止执行场景

- 接口签名：

```
public static void stopScene(Map<String, Object> params, ResponseCallback  
callback)
```

- 接口说明：停止执行场景

- 参数说明：

请求参数	{"id":"162257"}
响应结果	{

	<pre> "status": 0, "error": null, "result": { "sceneStatus": 3 // 1 完成, 2 执行中, 3 取消, 4 本地执行丢失的任务 } } </pre>
--	--

➤ 示例代码：

```

SceneManager.stopScene (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});

```

8.1.9 获取场景执行记录

➤ 接口签名：

```
public static void getSceneRecord (Map<String, Object> params, ResponseCallback callback)
```

➤ 接口说明：获取场景执行记录

➤ 参数说明：

请求参数	<pre> { "page": 1, // 第一页从 1 开始 "pageSize": 20 // 每页取的条数, 不传默认 10 条 } </pre>
响应结果	<pre> { "status": 0, "error": null, "result": [{ "recordId": 902893, // 执行记录 id }] } </pre>

	<pre> "name": "场景名称", "status": 3, // 1 完成, 2 执行中, 3 取消, 4 服务端丢失该场景执行 "startTime": "2018-05-09T09:26:59+0800", // 开始执行时间 "endTime": "2018-05-09T09:27:00+0800" // 结束执行时间 }, { "recordId": 902892, "name": "场景名称", "status": 1, "startTime": "2018-05-09T09:24:39+0800", "endTime": "2018-05-09T09:24:44+0800" }] } </pre>
--	---

➤ 示例代码：

```

SceneManager.getSceneRecord (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});

```

8.1.10 获取场景记录详情

➤ 接口签名：

```

public static void getSceneRecordDetail (Map<String, Object> params,
ResponseCallback callback)

```

➤ 接口说明：获取场景记录详情

➤ 参数说明：

请求参数	<pre> { "recordId": "902893" // 执行记录 id } </pre>
------	--

响应结果	<pre> { "status": 0, "error": null, "result": { "status": 3, // 1 完成, 2 执行中, 3 取消, 4 服务端丢失该场景执行 "exe_time": "2018-05-09T09:26:59+0800", "items": [{ "feed_id": 144309283668308867, "device_name": "古北智能插座(新)", "device_delete": 1, "status": 3, "startTime": "2018-05-09T09:26:59+0800", "streams": [{ "current_value": "1", "current_value_zh": "开", "stream_name_zh": "设置开关", "stream_id": "pwr", "real_value_zh": "" }] }, { "feed_id": 149796304011705303, "device_name": "ESP32_air_clean", "device_delete": 1, "status": 7, "startTime": "2018-05-09T09:27:00+0800", "streams": [{ "current_value": "1", "current_value_zh": "开", "stream_name_zh": "开关", "stream_id": "power", "real_value_zh": "" }] }] } } </pre>
------	--

	}
--	---

➤ 示例代码：

```

SceneManager.getSceneRecordDetail (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});

```

9 IFTTT

IFTTTManager IFTTT 管理器，主要提供了场景相关的京东云接口。详见 demo

9.1 IFTTT 云端接口

9.1.1 获取支持 IFTTT 的设备列表

➤ 接口签名：

```

public static void getIFTTTDeviceList(Map<String, Object> params,
ResponseCallback callback)

```

➤ 接口说明：获取支持 IFTTT 的设备列表

➤ 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

➤ 示例代码：

```

IFTTTManager.getIFTTTDeviceList (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {

```

```

    }
}
@Override
public void onFailure(String response) {
}
});

```

9.1.2 获取脚本列表

- 接口签名：

```
public static void getIFTTList (Map<String, Object> params, ResponseCallback
callback)
```

- 接口说明：获取脚本列表

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```

IFTTManager.getIFTTList (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
}
@Override
public void onFailure(String response) {
}
});

```

9.1.3 创建/修改脚本

- 接口签名：

```
public static void createScript (Map<String, Object> params, ResponseCallback
callback)
```

- 接口说明：创建/修改脚本

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTTManager.createScript (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

9.1.4 停用并删除脚本

- 接口签名：

```
public static void removeScript (Map<String, Object> params, ResponseCallback
callback)
```

- 接口说明：停用并删除脚本

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTTManager.removeScript (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

```
});
```

9.1.5 启动脚本

- 接口签名：

```
public static void startScript (Map<String, Object> params, ResponseCallback  
callback)
```

- 接口说明：启动脚本

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTManager.startScript (params, new ResponseCallback() {  
    @Override  
    public void onSuccess(String response) {  
        if (CommonUtil.isSuccess(response)) {  
        }  
    }  
    @Override  
    public void onFailure(String response) {  
    }  
});
```

9.1.6 停止脚本

- 接口签名：

```
public static void stopScript (Map<String, Object> params, ResponseCallback  
callback)
```

- 接口说明：停止脚本

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTManager.stopScript (params, new ResponseCallback() {
```

```

@Override
public void onSuccess(String response) {
    if (CommonUtil.isSuccess(response)) {
    }
}
@Override
public void onFailure(String response) {
}
});

```

9.1.7 检查脚本的状态

- 接口签名：

```
public static void checkStatus (Map<String, Object> params, ResponseCallback callback)
```

- 接口说明：检查脚本的状态

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```

IFTTManager.checkStatus (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});

```

9.1.8 手动执行场景

- 接口签名：

```
public static void scenarioActive (Map<String, Object> params, ResponseCallback
```

callback)

- 接口说明：手动执行场景
- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTTManager.scenarioActive (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});
```

9.1.9 获取场景执行日志

- 接口签名：
public static void getLog (Map<String, Object> params, ResponseCallback callback)
- 接口说明：获取场景执行日志
- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```
IFTTTManager.getLog (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
```



```

        public void onFailure(String response) {
        }
    });

```

9.1.10 获取用户场景执行记录

- 接口签名：

```

public static void getUserLogs (Map<String, Object> params, ResponseCallback
callback)

```

- 接口说明：获取用户场景执行记录

- 参数说明：

参数名	参数类型	必填	描述
params	Map<String, Object>	是	请求接口上行参数
callback	ResponseCallback	是	京东云接口回调

- 示例代码：

```

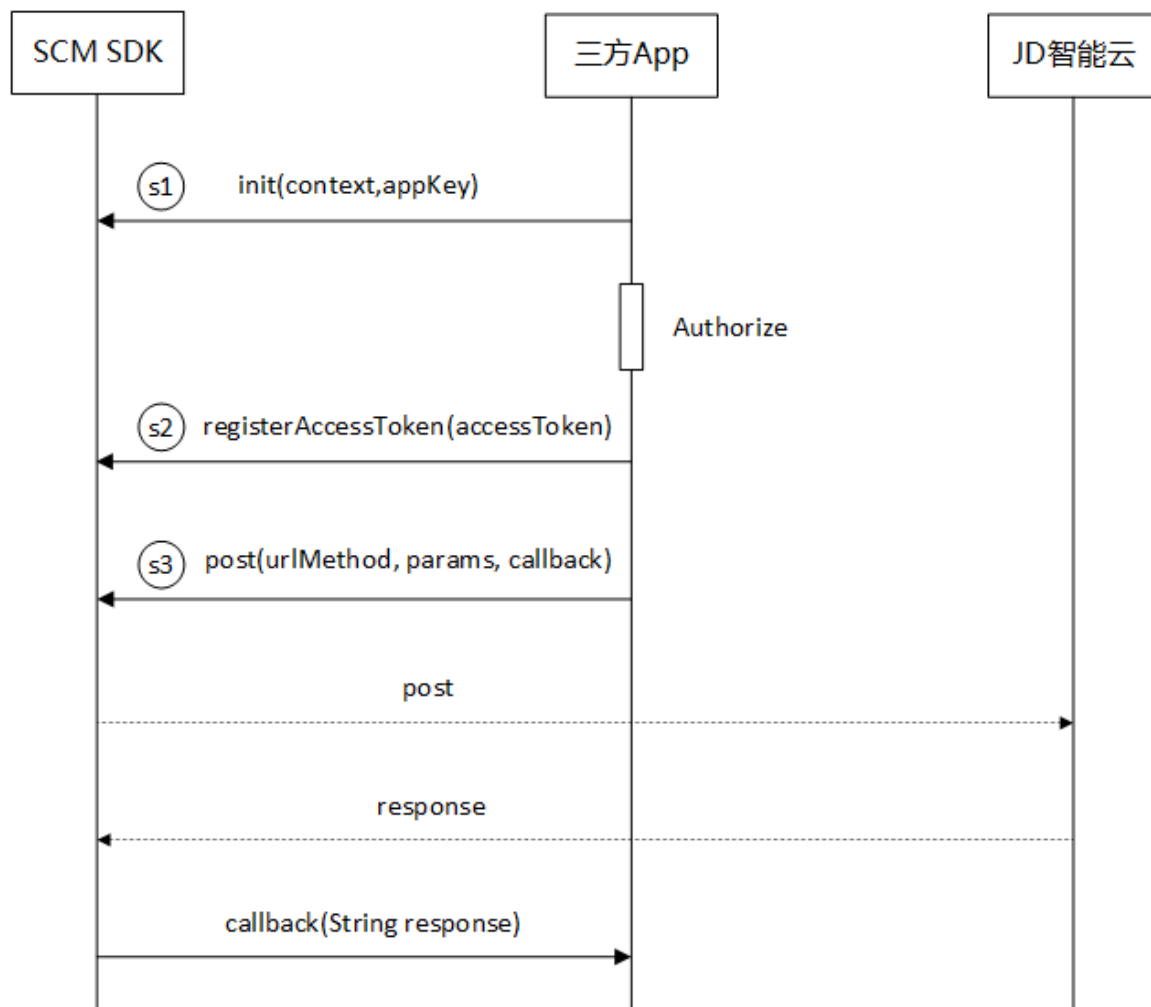
IFTTManager.getUserLogs (params, new ResponseCallback() {
    @Override
    public void onSuccess(String response) {
        if (CommonUtil.isSuccess(response)) {
        }
    }
    @Override
    public void onFailure(String response) {
    }
});

```

10 开放 API

OpenApiManager 开放 API 管理器，主要提供了访问云端开放 API 接口。详见 demo

10.1 时序图



10.2 post 请求

- 接口签名：
`public static void post(String urlMethod, String paramJson, ResponseCallback callback)`
- 接口说明：post 方式请求开放 API，详细的请求 API 接口方法和参数请咨询对接业务方。
- 参数说明：

参数名	参数类型	必填	描述
urlMethod	String	是	请求 API 接口方法
paramJson	String	是	请求参数，json 格式
callback	ResponseCallback	是	请求响应接口回调

- 示例代码：

```

OpenApiManager.post(urlMethod, paramJson, new ResponseCallback() {
    @Override

```

```

        public void onSuccess(String response) {
        }
        @Override
        public void onFailure(String response) {
        }
    });

```

11 相关错误码定义

编码	名称	消息	备注
1001	UNAUTHORIZED	无权限或未授权	
1002	ILLEGAL_PARAM	传入参数非法	
1003	ILLEGAL_USER	非法用户	
1004	REGISTER_FAILED	用户注册失败	
1005	BEENBIND	该设备已被绑定	
1006	BINDED	未找到数据	
2002	NOT_FOUND_PRODUCT_DATA	未找到产品数据	
2003	NOT_FOUND_DEVICE_DATA	未找到设备数据	
2004	DEVICE_STATUS_OFFLINE	设备不在线	
2005	SERVICE_UNAVAILABLE_BINDDEVICE_EXIST	设备已绑定	
2006	DEVICE_RES_TIMEOUT	设备响应超时	
2007	NO_SHARE_DEVICE_TOKEN	设备分享 TOKEN 不存在	
2008	CANNOT_SHARE_DEVICE	设备不可分享	
2009	NO_SHARE_DEVICE_RELATION	设备未分享给此用户	
2010	NOT_OWEN_DEVICE	用户对此设备没有分享权限	
2011	DEV_IN_IFTTT	设备参与了某个设备互联， 请先删除参与的设备互联	
2012	NOT_FOUND_ARTICLE_DATA	未找到文章信息	
2013	USER_BINDDEVICE_EXIST	设备未与用户绑定	
2014	NOT_FOUND_FIELD_DATA	上报数据字段信息未找到	
2015	HEALTH_USER_INIT_DATA	health_us、health_device 更新失败	
2017	DEVICE_ACTIVATE_ERROR	设备激活失败	
2018	DEVICE_BIND_ERROR	设备绑定失败	

2016	DEVICE_P2P_INFO_NOT_EXIST	设备 p2p 信息不存在	
2019	TEST_PRODUCT_EXPIRED	测试产品已过期	
2020	TEST_PRODUCT_COUNT_OVER	测试产品数量超过最大值	
2021	DEVICE_ACTIVATED_ERROR	设备已被激活	
2030	UNSUPPORTED_PLATFORM	暂不支持的客户端版本	
3001	SERVICE_UNAVAILABLE	服务不可用	
3002	SERVICE_UNAVAILABLE_PRODUCT_PUBLISHED	产品已发布,不更新	
3003	SERVICE_UNAVAILABLE_PRODUCT_EXIST	该产品名称和型号已存在	
3004	SERVICE_PROCESSING	设备繁忙,请稍后	
3005	SERVICE_UNAVAILABLE_DEVICE_EXIST	用户该设备名称已存在	
3006	NO_QR_CODE_FOUND	需先扫描二维码再激活设备	
5001	INTERNAL_SERVER_ERROR	网络异常	
9001	CALLBACK_ERROR	回调失败	
5002	HBASE_OPERATE_ERROR	Hbase 数据库错误	
6001	UNAUTHORIZED_PUBLIC_DEVICE	未授权公用设备,数据丢弃	
6002	SPORT_DATA_UPLOAD_ERROR	运动数据存储失败	
6003	TASK_DATA_UPLOAD_ERROR	任务数据存储失败	
7001	NO_CMD_FOUND	没有找到匹配的控制命令	
7002	RECOGNATION_COMMAND_CODE	识别到准确控制指令	
7003	HAS_SAME_TYPE_CODE	存在同类设备,您想操作哪一个呢?	
7004	RECOGNATION_QUERY_CODE	识别到查询结果	
7005	HAS_SAME_NAME_CODE	存在同名设备	
7006	RECOGNATION_DAILY_CONVERSATION_CODE	识别到日常对话	
7007	RECOGNATION_SCENE_CODE	识别到场景模式	
8001	THIRD_SERVICE_UNAVAILABLE	服务不可用(错误码:8001)	
9001	NOT_FIND_NEW_DEVICE	当前用户下,未找到新设备信息	