|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **设备配网FA开发指导书(个人开发者)** | | |  |
| **设备配网FA开发指导书(个人开发者)** | | |
| **文档版本** | **01** | |
| **发布日期** | **2021-05-25** | |
|  | | | | |
|  | **华为终端有限公司** | |  |  |

|  |
| --- |
| 版权所有 © 华为终端有限公司2021。 保留一切权利。  本材料所载内容受著作权法的保护，著作权由华为公司或其许可人拥有，但注明引用其他方的内容除外。未经华为公司或其许可人事先书面许可，任何人不得将本材料中的任何内容以任何方式进行复制、经销、翻印、播放、以超级链路连接或传送、存储于信息检索系统或者其他任何商业目的的使用。  商标声明  、、、华为，以上为华为公司的商标（非详尽清单），未经华为公司书面事先明示许可，任何第三方不得以任何形式使用。  注意  华为会不定期对本文档的内容进行更新。  本文档仅作为使用指导，文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为终端有限公司 | |
| 地址： | 广东省东莞市松山湖园区新城路2号 |
| 网址： | <https://consumer.huawei.com> |

目 录

[1 设备配网FA开发指导书 1](#_Toc256000000)

[1.1 开发指导 1](#_Toc256000001)

[1.2 参考信息 3](#_Toc256000002)

[1.3 配网接口调用 6](#_Toc256000003)

[1.3.1 是否持WifiAware 6](#_Toc256000004)

[1.3.2 获取Wifi列表 6](#_Toc256000005)

[1.3.3 通过NAN发现设备 7](#_Toc256000006)

[1.3.4 通过SoftAP发现设备 8](#_Toc256000007)

[1.3.5 连接设备 9](#_Toc256000008)

[1.3.6 对指定设备进行配网 11](#_Toc256000009)

[1.3.7 释放配网通道 13](#_Toc256000010)

[1.3.8 秒控接口 13](#_Toc256000011)

[1.3.9 获取设备侧消息 14](#_Toc256000012)

[1.3.10 设备断链通知回调消息 15](#_Toc256000013)

[1.4 常见问题 16](#_Toc256000014)

# 设备配网FA开发指导书

[1.1 开发指导](#_ZH-CN_TOPIC_0000001158765979)

[1.2 参考信息](#_ZH-CN_TOPIC_0000001111806146)

[1.3 配网接口调用](#_ZH-CN_TOPIC_0000001158605995)

[1.4 常见问题](#_ZH-CN_TOPIC_0000001158765981)

## 开发指导

概述

碰一碰设备控制模板（OneHop Device Feature Ability(JS) ）用于帮助开发者实现**“碰一碰-设备配网-设备控制”**完整流程。默认工程运行后为智能电风扇的样例效果，开发者可以参考[定义设备配网FA的UI界面](#section129531555424)章节修改成其他自定义设备。

该模板工程主要分为两部分：

**设备配网FA （entry模块）**

**设备控制FA （control模块）**

本指导书用于指导开发者如何开发**设备配网FA （entry模块）**，关于**设备控制FA（control模块）**的开发指导请参考设备控制FA开发指导书。

工程项目结构

OneHop Device Feature Ability(JS)碰一碰设备配网模板的工程目录结构如下所示。



源码目录中各个类的说明如下表所示。

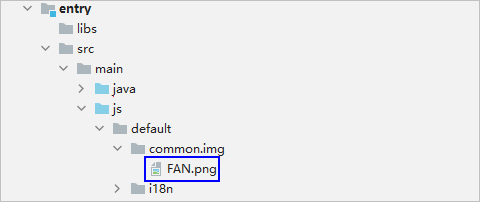
| Class | Description |
| --- | --- |
| **Java** | |
| MyApplication | 应用的入口类 |
| MainAbility | 拉起JS的入口类，处理NFC标签读取到的信息 |
| **JS** | |
| defualt/pages/device/device.js | 设备简介页，碰一碰拉起后跳转的页面 |
| default/pages/netconfig/netconfig.js | 设备配网页，负责配网相关逻辑，在设备简介页点击配网后显示 |

定义设备配网FA的UI界面

在MainAbility onStart里处理NFC标签读取到的product id信息， **开发者可以根据自己的productId映射成不同的设备类型，将productName修改成具体的设备名**，默认为智能电风扇。

@Override   
public void onStart(Intent intent) {   
 intent.setParam("window\_modal", 3);   
 setInstanceName(JS\_MODULE);   
   
 Object productInfo = Objects.requireNonNull(intent.getParams()).getParam("productInfo");   
 if (productInfo != null) {   
 productId = (String) productInfo;   
 // modify your product name according to your product id in nfc tag.   
 productName = "FAN";   
 }   
}

在资源目录下放置你的产品图， 用于界面显示，**注意文件命名为<设备名>.png， 设备名保持与productName一致**。



设备简介页和设备配网页会显示该产品图。

----结束

## 参考信息

关于工程中的核心代码说明如下：

1. 碰一碰拉起MainAbility后， 从intent中读取的nanSessionId用于配网，productId用于确定需要连接的设备类型，默认为智能电风扇。

参考MainAbility.java的onstart函数中如何读取设备信息。

同时需要将abilityContext和sessionId注册到配网SDK中。

String sessionId = intent.getStringParam("nanSessionId");   
if ("null".equals(sessionId) || "".equals(sessionId)) {   
 NetConfigAbility.register(this, null);   
} else {   
 NetConfigAbility.register(this, sessionId);   
}

1. MainAbility读取到的productId和productName会作为参数会传递到JS。

代码参考 MainAbility.java

IntentParams intentParams = intent.getParams();   
if (intentParams != null) {   
 intentParams.setParam("productId", productId);   
 intentParams.setParam("productName", productName);   
 setPageParams(null, intentParams);   
}

1. JS根据productName显示对应的设备图标， 代码参考device.js

onInit() {   
 this.deviceName = this.$t('strings.device-name');   
 this.deviceImg = '/common/img/' + this.productName + ".png";   
 getApp(this).Product.productName = this.productName;   
 getApp(this).Product.productId = this.productId;   
}

1. 点击配网后会执行配网逻辑，代码参考netconfig.js
   1. 通过NAN广播发现最近的一个设备

discoverDevice() {   
 let scanInfo = {   
 duration: 30,   
 lockTime: 60,   
 sessionId: ''   
 };   
 // Step1 discover the device through the NaN broadcast service.   
 getApp(this).NetConfig.discoveryByNAN(scanInfo, (result) => {   
 if (result.code == 0) {   
 this.progress = this.progressStatus[1];   
 this.desc = this.descStatus[1];   
 getApp(this).ConfigParams.deviceInfo = result.data;   
 this.registerDisconnectCallback(result.data.sessionId);   
 this.connectDevice();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[1];   
 this.disconnectDevice();   
 }   
 });   
}

* 1. 连接设备

connectDevice() {   
 let connectInfo = {   
 targetDeviceId: getApp(this).ConfigParams.deviceInfo.productId,   
 type: 0,   
 pin: '0123456789012345',   
 password: getApp(this).ConfigParams.deviceInfo.sn,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
 };   
 // Step2 connect the device.   
 getApp(this).NetConfig.connectDevice(connectInfo, (result) => {   
 if (result.code === 0) {   
 this.progress = this.progressStatus[2];   
 this.desc = this.descStatus[2];   
 this.getWifiInfo();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[2];   
 this.disconnectDevice();   
 }   
 });   
}

* 1. 获取手机连接的wifi信息

getWifiInfo() {   
 // Step3 get the WiFi information of the current phone connection.   
 getApp(this).NetConfig.getWifiList((result) => {   
 if (result.code == 0 && result.data && result.data.length > 0) {   
 this.progress = this.progressStatus[3];   
 this.desc = this.descStatus[3];   
 this.configDevice(result.data[0]);   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.descStatus[3];   
 this.disconnectDevice();   
 }   
 });   
}

* 1. 与设备进行配网， 配网成功后跳转到设备控制页面

async configDevice(wifiApInfo) {   
 let netConfigInfo = {   
 ssid: wifiApInfo.ssid,   
 ssidPassword: '',   
 isDefaultPassword: true,   
 channel: wifiApInfo.channel,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId,   
 type: 0,   
 wifiApId: wifiApInfo.wifiApId,   
 vendorData: '',   
 timeout: 30,   
 paramValid: true   
 };   
 // Step4 config the device net.   
 getApp(this).NetConfig.configDeviceNet('deviceInfo', 'accountInfo', netConfigInfo, (result) => {   
 if (result.code == 0) {   
 this.progress = this.progressStatus[4];   
 this.desc = this.descStatus[4];   
 this.registerMsgReceive()   
 // Step5 config the device net success, go to the control.   
 this.goToControl();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[4];   
 this.disconnectDevice();   
 }   
 });   
}



模板工程默认跳转到control设备控制模块，你也可以修改代码跳转到自定义的设备控制页，注意必须将sessionId传递到后续页面，用于给设备发送消息。

给设备发送消息的JS接口为：NetConfig.sendMessage(commonInfo, message, callbackFunc)

给设备发送消息的JAVA接口为：NetConfigApi.*getInstance*().sendMessage(CommonInfo, String, BaseCallback<String>)

## 配网接口调用

配网相关接口都在node\_modules/fa-netconfig.js下。

参考app.js中的import netConfig from 'fa-netconfig'进行引用。

### 是否持WifiAware

|  |  |
| --- | --- |
| 函数原型 | NetConfig.isSupportWifiAware() |
| 说明 | WifiAware是否支持。 |
| 接口类型 | 同步接口 |
| 返回值 | 0（支持）/ -1（不支持） |

* 参数

无

* 示例：

getApp(this).NetConfig.isSupportWifiAware().then(function (result) {   
 let ret = JSON.parse(result);   
 console.info("isSupportWifiAware消息结果: code=" + ret.code + ", data=" + ret.data);   
 if (ret.code == 0) {   
 //支持WifiAware   
 } else {   
 //不支持WifiAware   
 }   
});

### 获取Wifi列表

|  |  |
| --- | --- |
| 函数原型 | NetConfig.getWifiList(callbackFunc) |
| 说明 | 获取周边wifi列表。 |
| 接口类型 | 异步接口 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| callbackFunc | function | 是 | 回调函数，回调成功的情况下，data为返回数据结果，在该回调函数中解析使用数据结果。  返回：List<WifiApInfo>  WifiApInfo {  int channel;  boolean hasDefaultPassword;  boolean is5G; // 是否是5G wifi  String securityType;  int signalLevel;  String ssid;  String wifiApId; // wifiAp ID  }  2.4G和5G都返回，FA只需通过is5G是否支持5G进行筛选过滤；  周边范围内的  已配置过，并且能够成功获取密码的ssid，hasDefaultPassword才为true； |

* 示例

getApp(this).NetConfig.getWifiList((result) => {   
 if (result.code == 0 && result.data && result.data.length > 0) {   
 this.progress = this.progressStatus[3];   
 this.desc = this.descStatus[3];   
 this.configDevice(result.data[0]);   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.descStatus[3];   
 this.disconnectDevice();   
 }   
});

### 通过NAN发现设备

|  |  |
| --- | --- |
| 函数原型 | NetConfig.discoverDevByNAN(scanNanInfo, callbackFunc) |
| 说明 | 广播NAN服务，等待客户端连接，建立起一条连接，停止广播。 |
| 接口类型 | 异步接口 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| scanNanInfo | Object | 是 | scanNanInfo {  int duration; 在指定的duration时间内未建立起连接，停止广播，单位为秒，范围[0,100],0表示不限制时长。  int lockTime; 设备收到NAN互信消息后在NAN模式的锁定时长，单位为秒，范围[0,100],0表示不限制时长；新UX交互机制下，该字段功能暂不生效。  String sessionId; 值为service拉起FA时的Intent携带的sessionId值。  } |
| callbackFunc | function | 是 | 回调函数，回调成功的情况下，data为返回数据结果，在该回调函数中解析使用数据结果。返回：  DeviceInfo{  String productId,  String sessionId  String sn,  } |

* 示例：

let scanInfo = {   
 duration: 30,   
 lockTime: 60,   
 sessionId: ''   
};   
getApp(this).NetConfig.discoveryByNAN(scanInfo, (result) => {   
 if (result.code == 0) {   
 this.progress = this.progressStatus[1];   
 this.desc = this.descStatus[1];   
 getApp(this).ConfigParams.deviceInfo = result.data;   
 this.registerDisconnectCallback(result.data.sessionId);   
 this.connectDevice();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[1];   
 this.disconnectDevice();   
 }   
});

### 通过SoftAP发现设备

|  |  |
| --- | --- |
| 函数原型 | NetConfig.discoverDevBySoftAP(callbackFunc) |
| 说明 | 扫描AP。 |
| 接口类型 | 异步接口 |
| 输入参数 | 由FA自行确定调用次数 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| callbackFunc | function | 是 | 回调函数，返回数据结果，在该回调函数中解析使用数据结果。  返回：List<SoftAPInfo> SoftAPInfo{  String ssid,  boolean usePassword // 是否需要使用密码  } |

* 示例：

getApp(this).NetConfig.discoveryBySoftAP((result) => {   
 if (result.code == 0) {   
 for (let index = 0; index < result.data.length; index++) {   
 let element = result.data[index];   
 console.info("discoveryBySoftAP回调：device element=" + JSON.stringify(element));   
 // softAP扫描到的列表中，与既定的设备的ssid格式+品类+mac地址进行比对   
 if (element.ssid.indexOf(self.softAPSsidDefault) != -1) {   
 this.discoverDeviceBySoftAPResult = JSON.stringify(element);   
 this.softAPSsidDefault = element.ssid;   
 break;   
 }   
 }   
 }   
});

### 连接设备

|  |  |
| --- | --- |
| 数原型 | NetConfig.connectDevice(connectInfo,callbackFunc) |
| 说明 | 连接设备。 |
| 接口类型 | 异步接口 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| connectInfo | Object | 是 | connectInfo {  string targetDeviceId; //必填，待连接设备的标识。   * number type; //必填，（NAN）/ 1（SoftAp）。 * string pin; //必填，设备pin。 * string password; //必填，若设备需要密码，则设置为所需密码；否则设置为””。 * string sessionId; //必填，若type为NAN，则设置为discoverDevByNAN 返回的sessionId ；否则设置为“”。   } |
| callbackFunc | function | 是 | 回调函数，回调成功的情况下，data为返回数据结果，在该回调函数中解析使用数据结果。  返回：String vendorData // 后续扩展使用。  Softap模式下返回{  String productId，  String sn，  String vendorData  } 的JSON格式字符串 |

* NAN类型示例

let connectInfo = {   
 targetDeviceId: getApp(this).ConfigParams.deviceInfo.productId,   
 type: 0,   
 pin: '0123456789012345',   
 password: getApp(this).ConfigParams.deviceInfo.sn,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
};   
getApp(this).NetConfig.connectDevice(connectInfo, (result) => {   
 if (result.code === 0) {   
 this.progress = this.progressStatus[2];   
 this.desc = this.descStatus[2];   
 this.getWifiInfo();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[2];   
 this.disconnectDevice();   
 }   
});

* SoftAP类型示例

let connectInfo = {   
 targetDeviceId: getApp(this).ConfigParams.deviceInfo.productId,   
 type: 1,   
 pin: '0123456789012345',   
 password: getApp(this).ConfigParams.deviceInfo.sn,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
};   
getApp(this).NetConfig.connectDevice(connectInfo, (result) => {   
 if (result.code === 0) {   
 this.progress = this.progressStatus[2];   
 this.desc = this.descStatus[2];   
 this.getWifiInfo();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[2];   
 this.disconnectDevice();   
 }   
});

### 对指定设备进行配网

|  |  |
| --- | --- |
| 函数原型 | NetConfig.configDeviceNet(deviceInfo,accountInfo,netConfigInfo,callbackFunc) |
| 说明 | 开始对指定的设备进行配网，发送SSID和密码，以及设备和账号信息。 |
| 接口类型 | 异步接口 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| deviceInfo | string | 是 | 设备云为设备分配的信息，如deviceID等 |
| accountInfo | string | 是 | 账号信息 |
| netConfigInfo | Object | 是 | netConfigInfo {  string ssid;  //必填，如果是手机已经配置过的ssid，可免输密码。  string ssidPassword;//非必填，不需要下发密码时，填写“”。  boolean isDefaultPassword; //必填。  int channel; //必填。  string sessionId; //必填,type为NAN时填实际值；type为softAp时填“”。  int type; //必填,0（NAN）/ 1(softAp)。  String wifiApId; //填写wifi列表返回信息中的对应字段。  String vendorData; //产品数据，当前为random Base64编码;  int timeout; // 配网超时时间，在NAN模式下设置，范围[1,90]s。  }  说明:如果用户输入密码，则isDefaultPassword=false, ssidPassword=“输入的密码”；否则按使用默认密码处理，isDefaultPassword=true, ssidPassword=”” |
| callbackFunc | function | 是 | 回调函数，NAN配网模式中，回调成功表示模组上报配网状态结果，其结果会转换成整型数值放到data.code，字符串类型会放到data.msg; SoftAp配网模式中，返回0/ -1表示  配网成功/失败； |

* NAN类型示例

let netConfigInfo = {   
 ssid: wifiApInfo.ssid,   
 ssidPassword: '',   
 isDefaultPassword: true,   
 channel: wifiApInfo.channel,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId,   
 type: 0,   
 wifiApId: wifiApInfo.wifiApId,   
 vendorData: '',   
 timeout: 30,   
 paramValid: true   
};   
getApp(this).NetConfig.configDeviceNet('deviceInfo', 'accountInfo', netConfigInfo, (result) => {   
 if (result.code == 0) {   
 this.progress = this.progressStatus[4];   
 this.desc = this.descStatus[4];   
 this.registerMsgReceive()   
 this.goToControl();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[4];   
 this.disconnectDevice();   
 }   
});

* SoftAP类型示例

let netConfigInfo = {   
 ssid: wifiApInfo.ssid,   
 ssidPassword: '',   
 isDefaultPassword: true,   
 channel: wifiApInfo.channel,   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId,   
 type: 1,   
 wifiApId: wifiApInfo.wifiApId,   
 vendorData: '',   
 timeout: 30,   
 paramValid: true   
};   
getApp(this).NetConfig.configDeviceNet('deviceInfo', 'accountInfo', netConfigInfo, (result) => {   
 if (result.code == 0) {   
 this.progress = this.progressStatus[4];   
 this.desc = this.descStatus[4];   
 this.registerMsgReceive()   
 this.goToControl();   
 } else {   
 this.progress = this.progressStatus[4];   
 this.desc = this.failDescStatus[4];   
 this.disconnectDevice();   
 }   
});

### 释放配网通道

|  |  |
| --- | --- |
| 函数原型 | NetConfig.disconnectDevice(commonInfo,callbackFunc) |
| 说明 | 释放配网通道，在FA不再使用NAN通道、FA退出等情况需要FA主动调用该接口。 |
| 接口类型 | 异步接口 |
| 输入参数 |  |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| commonInfo | Object | 是 | commonInfo{  String sessionId; //若type为NAN，则设置为discoverDevByNAN 返回的sessionId ；SoftAp设置为””。  } |
| callbackFunc | function | 是 | 回调函数，返回数据结果，在该回调函数中解析使用数据结果。  返回：0（成功）/ -1（失败） |

* 示例：

let commonInfo = {   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
}   
getApp(this).NetConfig.disconnectDevice(commonInfo, (result) => {   
 if (result.code == 0) {   
 return;   
 }   
});

### 秒控接口

|  |  |
| --- | --- |
| 函数原型 | NetConfig.sendMessage(commonInfo,message,callbackFunc) |
| 说明 | 秒控接口。 |
| 接口类型 | 异步接口 |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| commonInfo | Object | 是 | commonInfo{  String sessionId; //若type为NAN，则设置为discoverDevByNAN 返回的sessionId ；否则设置为””。  } |
| message | String | 是 | 消息内容 |
| callbackFunc | function | 是 | 回调函数，返回命令发送结果。  返回：0（成功）/ -1（失败） |

* 示例：

let commonInfo = {   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
}   
getApp(this).NetConfig.sendMessage(commonInfo, "111111", (result) => {   
 // sendMessage 回调   
})

### 获取设备侧消息

|  |  |
| --- | --- |
| 函数原型 | NetConfig.registerMsgReceive(commonInfo, callbackFunc) |
| 说明 | 获取设备侧消息 //建议在NAN配网模式下连接设备接口前调用。 |
| 接口类型 | 异步接口 |
| 输入参数 |  |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 | |
| commonInfo | Object | 是 | commonInfo{  String sessionId; //若type为NAN，则设置为discoverDevByNAN 返回的sessionId ；否则设置为””。  } |  |
| callbackFunc | function | 是 | 回调函数，回调成功的情况下，data为返回数据结果，在该回调函数中解析使用数据结果。  返回：String message |  |

* 示例：

let commonInfo = {   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
}   
getApp(this).NetConfig.registerMsgReceive(commonInfo, () => {   
 // registerMsgReceive 回调   
});

### 设备断链通知回调消息

|  |  |
| --- | --- |
| 函数原型 | NetConfig.registerDisconnectCallback(commonInfo, callbackFunc) |
| 说明 | 注册连接断开事件回调，建议在NAN配网模式下连接设备接口前调用。  当因某些原因造成设备主动与手机断开连接时，会通过该回调通知。 |
| 接口类型 | 异步接口 |
| 输入参数 |  |
| 返回值 | 0（成功）/ -1（失败） |

* 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必填 | 说明 |
| commonInfo | Object | 是 | commonInfo{  String sessionId; //若type为NAN，则设置为discoverDevByNAN 返回的sessionId ；否则设置为””。  } |
| callbackFunc | function | 是 | 回调函数，当收到该回调时表示设备与手机连接断开。  返回状态码：0。 |

* 示例

let commonInfo = {   
 sessionId: getApp(this).ConfigParams.deviceInfo.sessionId   
}   
getApp(this).NetConfig.registerDisconnectCallback(commonInfo, () => {   
 // registerDisconnectCallback 回调   
});

## 常见问题

1. **应用退出后重新进控制页面，不能继续控制设备？**

目前每次进入应用都要先配网获取新的控制通路，同时配网前需要复位设备，否则设备侧会维持上一次配网状态而无法重新配网。

1. **是否必须碰一碰才能进入配网功能？**

直接打开entry的设备配网页也可以配网，碰一碰NFC标签只是用于拉起你的FA应用，并且通过NFC信息获取到碰一碰接触的设备信息。

1. **点击配网后，配网进度卡在20%之后突然显示失败 ？**

由于设备侧配网版本可能不太稳定，这种情况可能会发生在上一次应用页面被杀死或者异常退出的时候。

如果出现配网失败， 可以尝试退出应用，并且复位设备后再尝试。