青莲云 iOS 蓝牙 SDK 使用文档

版本记录：

版本	编写/修订说明	编写人	编写日期	备注
1.0.0	编写文档		20190508	本地蓝牙
2.0.0	1、新增云端蓝牙的注册 SDK。 2、新增云端蓝牙搜索设备、连接设备等功能。新增设备 OTA 升级（下载和传输）及取消升级的功能。 3、新增用户管理功能。包括注册、登录、找回密码、获取用户信息、获取登录状态、退出登录功能。 4、新增产品管理功能。包括获取产品列表、获取真实 productKey、获取产品对象、获取上报数据点、获取下发数据点。 5、新增设备管理功能。包括获取用户绑定设备列表、获取设备详细信息、修改设备名称、检查用户绑定设备是否可升级、检查用户绑定设备的 OTA 升级状态。		20190524	添加云端功能
2.1.0	更新 OTA 升级相关的方法。 1、新增 OTA 固件后台传输及断点续传的功能。 2、新增查询某设备当前 OTA 传输状态方法。		20190712	更新 OTA 升级功能
3.0.0	新增蓝牙 Mesh 的相关方法。 1、创建一个 Mesh 网络。 2、删除一个 Mesh 网络。 3、获取所有 Mesh 网络。 4、连接 Mesh 网络。 5、断开 Mesh 网络。 6、获取 Mesh 网络的所有节点 7、修改网络的名称及 TTL。 8、搜索周围的可入网节点。 9、Mesh 网络添加节点。 10、重置节点。 11、删除节点信息。 12、修改节点名称。 13、向节点发送二进制透传消息。		20190918	添加蓝牙 Mesh 功能

	14、向节点发送数据点消息。 15、获取节点的订阅节点列表。 16、让节点去订阅另一个节点。 17、让节点取消订阅某个节点。 18、向节点下发极速小数据点。 19、实时接收节点发送二进制透传。 20、实时接收节点上报数据点。 21、实时接收节点上报极速小数据点消息。 22、监听网络连接和断开。			
4.0.0	新增蓝牙 Mesh 和云端交互功能的相关方法。 1、创建 Mesh 网络并同步到云端。 2、删除指定的 Mesh 网络。 3、获取所有 Mesh 网络。 4、连接 Mesh 网络。 5、断开 Mesh 网络。 6、获取 Mesh 网络的所有节点 7、修改网络的名称及 TTL。 8、搜索周围的可入网节点。 9、Mesh 网络添加节点。 10、重置节点。 11、清除节点信息。 12、修改节点名称。 13、向节点发送二进制透传消息。 14、向节点发送数据点消息。 15、获取节点的订阅节点列表。 16、让节点去订阅另一个节点。 17、让节点取消订阅某个节点。 18、向节点下发极速小数据点。 19、实时接收节点发送二进制透传。 20、实时接收节点上报数据点。 21、实时接收节点上报极速小数据点消息。 22、监听网络连接和断开。		20191108	添加蓝牙 Mesh 和云端交互的对应功能。

目录

一、概要

青莲云作为物联网后端云服务，用户可以使用青莲云提供的 iOS_BLE_SDK 快速进行 APP 蓝牙功能的开发。青莲云 iOS_BLE_SDK 提供了与硬件设备及云端通讯的接口封装，主要包括以下三种类型：

1、本地蓝牙

本地蓝牙功能可提供 APP 与设备交互的相关接口，主要包括以下功能：

扫描周围设备、连接设备、断开连接、发送透传消息、发送数据点信息、发送解绑命令、实时监听设备非正常连接状态、实时接收设备透传消息、实时接收设备上传的数据点信息、监听设备主动解绑消息。

2、云端蓝牙

云端蓝牙功能除提供以上 APP 与设备交互以外还拥有用户、产品、OTA 升级相关功能。

3、蓝牙 Mesh

蓝牙 Mesh 是种基于 BLE 蓝牙的自组网技术。此 SDK 中的蓝牙 Mesh 相关功能包括：

创建一个 Mesh 网络、删除一个 Mesh 网络、获取所有 Mesh 网络、连接 Mesh 网络、断开当前 Mesh 网络、获取当前 Mesh 网络的所有节点、修改网络的名称及 TTL、搜索周围的可配网节点、给 Mesh 网络添加节点、重置节点、删除节点信息、修改节点名称、向节点发送二进制透传消息、向节点发送数据点消息、获取节点的订阅节点列表、让节点去订阅另一个节点、让节点取消订阅某个节点、实时接收节点二进制透传消息、实时接收节点上报数据点、监听 Mesh 网络连接及断开。

4、云端蓝牙 Mesh

云端蓝牙 Mesh 是在蓝牙 Mesh 的基础上加入了云端功能，将 Mesh 网络的信息同步到云端，同时 Mesh 网络及网络内的节点都将和用户进行绑定，因此您可在不同手机甚至不同的平台（安卓/iOS）上登录您的账号即可获取您的账号绑定 Mesh 网络的信息及绑定节点的信息。前提条件必须要导入 IOTSDK.framework 最为支撑。

二、SDK 介绍

1、SDK 名称

SDK 名称：IOTBlueToothSDK.framework

SDK 支持的 iOS 版本：iOS 10.0 及以上。

2、接入说明

下载 IOTBlueToothSDK.framework，将该文件拷贝到工程目录下即可。（友情提示：如果出现库文件不能正确导入，可尝试新建 New Copy Files Phase，选择 Frameworks，添加 IOTBlueToothSDK.framework）

3、注意事项

若只使用本地蓝牙功能则只需下载 IOTBlueToothSDK.framework 导入项目即可,若需要使用云端蓝牙或云端蓝牙 Mesh 相关功能则还需要从青莲云官网下载 IOTSDK.framework 作为支撑。

使用蓝牙 Mesh（云端&非云端）相关功能时需在工程中创建一个任意的 swift 文件并生成相应的桥接文件，否则会出现一些 swift 系统库无法找到的问题。

三、开发前准备

前往 Targets --> Build Settings --> Other Linker Flags 添加-ObjC，否则会出现相关方法找不到导致崩溃。

1、SDK 初始化

在使用此 SDK 之前需要使用相关信息进行 SDK 的初始化，否则 SDK 将不可用。SDK 初始化步骤如下：

1、在项目 TAGETS —> General —> Linked Frameworks and Libraries 下点击加号，加入 IOTBlueToothSDK.framework。

2、在项目的预编译文件 PrefixHeader.pch 中添加如下内容：

```
#import <IOTBlueToothSDK/IOTBlueToothSDK.h>
```

3、打开项目的 AppDelegate.m 文件，在 application:didFinishLaunchingWithOptions:方法中使用 IOTSmartBlueToothSDK 中的 IOTCloudSDK_startAPPWithCompanyID:companyToken:AppID:AppToken:APIHost:FileHost:isDebug:success:failure:方法进行 SDK 的初始化。

4、注意事项：

若应用中不会使用到 IOTSmartCloudBlueTooth 或 IOTSmartCloudBlueToothMesh 类，即不会使用云端蓝牙及云端蓝牙 Mesh 功能，则 companyId,companyToken,appId,appToken,apiHost,fileHost 参数都传 nil 即可。

若应用中会使用到云端功能，则此方法的所有参数都不可为 nil，且需要从青莲云开放平台下载 IOTSDK.framework 导入项目。在项目 TAGETS —> General —> Embedded binaries 下点击加号，添加 IOTSDK.framework。同时需要在项目的预编译文件 PrefixHeader.pch 中

添加 **#import<IOTSDK/IOTSDK.h>**。

请求参数：

参数名	类型	说明	备注	必需
companyId	String	公司 Id	青莲云云平台获取	
companyToken	String	公司 Token	青莲云云平台获取	
appId	String	AppId	青莲云云平台获取	
appToken	String	AppToken	青莲云云平台获取	
apiHost	String	接口服务器域名地址	贵公司申请的域名	
fileHost	String	文件服务器域名地址	贵公司申请的域名	

例子：

```
[[IOTSmartBlueToothSDK shareInstance]
IOTCloudSDK_startAPPWithCompanyId:@"companyId" companyToken:@"companyToken"
appId:@"appId" appToken:@"appToken" apiHost:@"apiHost" fileHost:@"fileHost "
isDebug:@"isDebug" success:^{
   NSLog (@"start success");
} failure:^(int errNo, NSString *errMessage) {
   NSLog (@"start failure");
}];
```

四、SDK 使用

1、本地蓝牙

本地蓝牙相关的所有功能对应 IOTSmartBlueTooth 类（单例）。

1.1、搜索设备

可选择搜索全部设备和通过设备名称进行搜索，当 FuzzyName 设置为 nil 时代表搜索全部设备，每搜索到一个设备就会调用一次 success 回调。

请求参数：

参数名	类型	说明	备注	必需
FuzzyName	String	设备名称（可为 nil）	模糊搜索设备	否
timeout	int	搜索的超时时间	无	是

例子：

```
- (void)searchDevice{
```

```
    [[IOTSmartBlueTooth shareInstance] IOTCloudSDK_searchWithFuzzyName:
@"fuzzy_device_name" timeout: @"timeout" success:^(NSString * _Nonnull peripheralUUID,
NSString * _Nonnull peripheralName) {
        NSLog (@"search_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"search_failure");
    }];
}
```

1.2、连接设备

连接搜索到的设备，若未进行搜索，则 SDK 会默认执行一次搜索操作。连接成功即与设备绑定。若设备已连接，则直接返回连接成功。

请求参数：

参数名	类型	说明	备注	必需
peripheralUUID	String	设备的 uuid	无	是
userId	int	用户 Id	无	是
productId	int	设备所属产品的 Id	与相应设备烧录时写入的一样。	是
productKey	String	设备所属产品的 Key	16 个字节的十六进制字符串，与相应设备烧录时写入的一样。	是

例子：

```
- (void)connDevice{
    [[IOTSmartBlueTooth shareInstance] IOTCloudSDK_connectWithPeripheralUUID:
@"peripheral_uuid" userId:@"your_userId" productId:@"your_productId" productKey:
@"your_productKey" success:^{
        NSLog (@"connection_device_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"connection_device_failure");
    }];
}
```

1.3、断开连接

断开已经连接的设备。若设备原本就未连接，则直接返回断开成功。

请求参数：

参数名	类型	说明	备注	必需

peripheralUUID	String	设备的 uuid	无	是

例子：

```
- (void)disConnection{
    [[IOTSmartBlueTooth shareInstance]
IOTCloudSDK_disConnectWithPeripheralUUID:@"peripheral_uuid" success:^ {
        NSLog (@"disConnection_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"disConnection_failure");
    }];
}
```

1.4、发送透传消息

连接设备后才能给设备发送二进制透传消息。

请求参数：

参数名	类型	说明	备注	必需
peripheralUUID	String	设备的 uuid	无	是
publishData	NSData	要发送给设备的数据	无	是

例子：

```
- (void)pushData {
    [[IOTSmartBlueTooth shareInstance] IOTCloudSDK_publishData:@"pushData"
peripheralUUID: @"peripheral_uuid" success:^ {
        NSLog (@"publishData_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"publishData_failure");
    }];
}
```

1.5、下发数据点信息

连接设备后能够向设备发送数据点信息。数据点相关信息请参见 IOTBlueToothSDK.framework 中的 IOTSmartBlueToothDPModel 类。

请求参数：

参数名	类型	说明	备注	必需
peripheralUUID	String	设备的 uuid	无	是
dpArr	NSArray	存放数据点对象的数组	只能存放数据点对象	是

例子：

```
- (void)pushDps {
```

```
    [[IOTSmartBlueTooth shareInstance] IOTCloudSDK_publishDps: @"dpArr "
peripheralUUID: @"peripheral_uuid" success:^{
        NSLog (@"publishDps_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"publishDps_failure");
    }];
}
```

1.6、发送解绑命令

app 端主动解除设备与用户的绑定关系。前置条件是需要和设备先建立连接。

请求参数：

参数名	类型	说明	备注	必需
peripheralUUID	String	设备的 uuid	无	是

例子：

```
- (void)unbind {
    [[IOTSmartBlueTooth shareInstance]
IOTCloudSDK_unbindWithPeripheralUUID:@"peripheral_uuid" success:^{
        NSLog (@"unbind_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"unbind_failure");
    }];
}
```

1.7、实时获取设备透传消息

实现 IOTSmartBlueToothDelegate 的 IOTCloudSDK_peripheral:pushData:方法即可实时获取设备上传的二进制透传数据。

1.8、实时获取设备上报数据点

实现 IOTSmartBlueToothDelegate 的 IOTCloudSDK_peripheral:uploadDps:方法即可实时获取设备发送的数据点对象的数组。

1.9、监听设备非正常断开及自动重连成功

当设备非正常断开后 SDK 会自动进行重连操作。手机端要监听设备非正常断开及自动重连成功。需将自身作为 IOTSmartDeviceConnectionStatus 通知的观察者，当设备非正常断开或 SDK 自动重连成功时 SDK 会给每个观察者发送通知，具体通知消息可从通知对象的 userInfo 中获取。（非正常断开：除了 APP 调用 SDK 提供的断开连接方法及设备主动解绑以外的所有原因导致的断开连接，都被认为是非正常断开）。

1.10、监听设备解除绑定

手机端要监听设备解绑消息。需将自身作为 IOTSmartDeviceLaunchUnbind 通知的观察者。当设备主动发起解绑时会提示用户已被解绑。注意：设备主动解绑是正常断开连接，SDK 不会自动进行重连。

2、云端蓝牙

云端蓝牙拥有本地蓝牙的所有功能，并增加了蓝牙设备 OTA 升级相关的方法。其相应功能请参见 IOTSmartCloudBlueTooth 类（单例）。

2.1、搜索设备

根据外设名称模糊搜索外部设备,要搜索全部可将 deviceFuzzyName 设置为@""，每搜索到一个设备就会调用一次 success 回调，返回设备 UUID，设备名称，设备 Mac 地址。

请求参数：

参数名	类型	说明	备注	必需
FuzzyName	String	设备名称（可为 nil）	模糊搜索设备	否
timeout	int	搜索的超时时间	无	是

例子：

```
- (void)searchDevice{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_searchWithFuzzyName:@"fuzzy_name " timeout: @"timeout" success: ^(NSString *peripheralUUID, NSString *peripheralName, NSString *peripheralMac){
            NSLog (@"search_success");
    } failure:^(NSString * _Nonnull errMessage) {
            NSLog (@"search_failure");
    }];
}
```

2.2、连接设备

连接搜索到的设备，若未进行搜索，则 SDK 会默认执行一次搜索操作。连接成功即与设备绑定并向云端提交绑定信息。若设备已连接，则直接返回连接成功。

请求参数：

参数名	类型	说明	备注	必需
mac	String	设备的 mac	无	是
productId	int	设备所属产品的 Id	无	是
productKey	String	设备所属产品的 Key	无	是

例子：

```
- (void)connDevice{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_connectWithPeripheralMac:@"peripheral_mac" productId:@" productId" productKey: @" productKey" success:^{
        NSLog (@"connection_device_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"connection_device_failure");
    }];
}
```

2.3、断开连接

断开已经连接的设备。若设备原本就未连接，则直接返回断开成功。

请求参数：

参数名	类型	说明	备注	必需
mac	String	设备的 mac	无	是

例子：

```
- (void)disConnection{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_disConnectWithMac:@"peripheral_mac" success:^ {
        NSLog (@"disConnection_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"disConnection_failure");
    }];
}
```

2.4、发送透传消息

连接设备后能够给设备发送二进制透传消息。即 NSData 类型的数据。

请求参数：

参数名	类型	说明	备注	必需
mac	String	设备的 mac	无	是
publishData	NSData	要发送给设备的数据	无	是

例子：

```
- (void)pushData {
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_publishData:@"pushData" mac:@"peripheral_mac" success:^ {
        NSLog (@"publishData_success");
```

```
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"publishData_failure");
    }];
}
```

2.5、下发数据点信息

连接设备后能够向设备发送数据点信息。数据点相关信息请参见 IOTBlueToothSDK.framework 中的 IOTSmartBlueToothDPModel 类。

请求参数：

参数名	类型	说明	备注	必需
mac	String	设备的 mac	无	是
dpArr	NSArray	存放数据点对象的数组	只能存放数据点对象	是

例子：

```
- (void)pushDps {
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_publishDps: @"dpArr " mac:@"peripheral_mac" success:^{
        NSLog (@"publishDps_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"publishDps_failure");
    }];
}
```

2.6、发送解绑命令

APP 端主动解除设备与用户的绑定关系。前置条件是需要和设备先建立连接。

请求参数：

参数名	类型	说明	备注	必需
mac	String	设备的 mac	无	是

例子：

```
- (void)unbind {
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_unbindWithMac:@"peripheral_mac" success:^{
        NSLog (@"unbind_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"unbind_failure");
    }];
}
```

2.7、设备 OTA 升级

设备的 OTA 升级过程可分为三步：一是下载 OTA 升级固件，二是向指定设备传输升级固件数据包，三是重启设备并立即升级。

2.7.1、下载 OTA 升级固件

APP 在联网的情况下可从云端下载某设备最新版本的固件数据包到本地缓存中。

请求参数：

参数名	类型	说明	备注	必需
productId	int	设备所属产品的产品 Id	无	是
iotId	String	要下载新版本的设备的 iotId	无	是
newVersion	String	要下载固件的版本号	无	是
deviceType	int	需要下载的固件类型	1:mcu 2:wifi 3:蓝牙	是

例子：

```
- (void)download{
    [[IOTSmartCloudBlueTooth shareInstance]
IOTCloudSDK_downloadDeviceNewVersionWithProductId: @"productId" iotId: @"iotId"
newVersion: @"newVersion" deviceType: @"deviceType" success:^{
        NSLog (@"download_success");
    } failure:^(NSString * _Nonnull errMessage) {
        NSLog (@"download_failure");
    }];
}
```

2.7.2、开始传输 OTA 升级固件

APP 开始向设备传输 OTA 升级包，开启成功会收到成功的回调并可通过监听相关的通知实时获取传输进度。注意：必须先连接上该设备才能进行传输。

请求参数：

参数名	类型	说明	备注	必需
mac	String	要升级的设备的 mac	无	是
newVersion	String	要升级的版本号	无	是
deviceType	int	需要升级的固件类型	1:mcu 2:wifi 3:蓝牙	是

例子：

```
- (void) startTransfer{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_startTransferOTADataWithMac:
@"mac" newVersion: @"newVersion" deviceType: @"deviceType" success: ^{
        NSLog (@"startTransferOTAData");
    } failure:^(NSString * _Nonnull errMessage) {
```

```
            NSLog (@"start_failure");
        }];
}
```

2.7.3、立即升级

当设备 OTA 传输完成后可调用此方法使设备重启并升级新版本。

请求参数：

参数名	类型	说明	备注	必需
mac	String	要升级的设备的 mac	无	是
newVersion	String	要升级的版本号	无	是
deviceType	int	需要升级的固件类型	1:mcu 2:wifi 3:蓝牙	是
timeout	int	升级的超时时间	单位：秒	是

例子：

```
- (void) startUpgrade{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_upgradeImmediatelyWithMac: @"mac" newVersion: @"newVersion" deviceType: @"deviceType" timeout: @"timeout" success: ^{
            NSLog (@"upgrade success");
    } failure:^(NSString * _Nonnull errMessage) {
            NSLog (@"upgrade failure");
    }];
}
```

2.8、取消 OTA 传输或暂不升级

在进行 2.7.2 的 OTA 固件传输过程中可调用此方法取消 OTA 传输，或当 OTA 固件传输完成后用户暂时不想重启设备时调用。

请求参数：

参数名	类型	说明	备注	必需
mac	String	要升级的设备的 mac	无	是
deviceType	int	设备的类型	无	是

例子：

```
- (void)cancelUpgrade{
    [[IOTSmartCloudBlueTooth shareInstance] IOTCloudSDK_pauseOTAUpgradeWithMac: @"mac" deviceType: @"deviceType"];
}
```

2.9、获取某设备当前的 OTA 传输状态

此方法可获取到某个设备当前的 OTA 传输状态，有三种状态：未传输，传输中，传输完成。详情请见枚举类型 IOTSmartCloudOTAProgress。

请求参数：

参数名	类型	说明	备注	必需
mac	String	查询传输状态的设备的 mac	无	是

例子：

```
- (NSInteger)getTransStatus {
    NSInteger state = [[IOTSmartCloudBlueTooth shareInstance]
IOTCloudSDK_getOTAProgressWithMac:@"mac"];
    return state;
}
```

2.10、实时获取设备的透传消息

实现 IOTSmartCloudBlueToothDelegate 的 IOTCloudSDK_mac: pushData:代理方法即可实时获取设备发送的透传信息。

2.11、实时获取设备上报数据点

实现 IOTSmartCloudBlueToothDelegate 的 IOTCloudSDK_mac:uploadDps:代理方法即可实时获取设备上报的数据点数据。

2.12、监听设备非正常断开及自动重连成功

当设备非正常断开后 SDK 会自动进行重连操作。手机端要监听设备非正常断开及自动重连成功，需将自身作为 IOTSmartDeviceCloudConnectionStatus 通知的观察者，当设备非正常断开或 SDK 自动重连成功时 SDK 会给每个观察者发送通知，具体通知消息可从通知对象的 userInfo 中获取。（非正常断开：除了 APP 调用 SDK 提供的断开连接方法及设备主动解绑以外的所有原因导致的断开连接，都被认为是非正常断开）。

2.13、监听设备主动解绑

手机端要监听设备解绑消息。需将自身作为 IOTSmartDeviceCloudLaunchUnbind 通知的观察者。当设备主动发起解绑时会提示用户已被解绑。注意：设备主动解绑是正常断开连接，SDK 不会自动进行重连。

2.14、监听设备 OTA 传输进度

手机端要监听设备 OTA 传输的进度，需将自身作为 IOTSmartDeviceCloudOTAProgress 通知的观察者。当设备进行 OTA 固件传输时可实时获取到设备的传输进度等数据。

3、用户管理

用户相关的所有功能对应 IOTSDK.framework 中的 IOTSmartUser 类（单例）。

3.1、注册用户

用户可使用自己的手机号注册账号，注册过程分为两步：一是发送验证码到手机，二是通过验证码注册账号。注册成功即为登录成功，不需要再次调用登录方 法。

3.1.1、发送注册验证码到手机

此短消息模版是青莲云注册用户消息模版。

请求参数：

参数名	类型	说明	备注	必需
phone	String	手机号码	接收验证码的手机号	是
zone	String	区号（如中国大陆手机号写“0086”）	默认四位，不足四位前面补 0	是

例子：

```
- (void)sendVerifyCode {
[[IOTSmartUser sharedInstance] IOTCloudSDK_registerSendVerifyCodeByPhoneNum:
@"your_phone_number" zone: @"your_phone_zone"  success:^{
        NSLog(@"sendVerifyCode success");
} failure:^(int errNo,NSString *errMessage) {
        NSLog(@"sendVerifyCode failure: %@", errMessage);
}];
}
```

3.1.2、使用手机验证码注册

请求参数：

参数名	类型	说明	备注	必需
phone	String	手机号码	接收验证码的手机号	是
zone	String	区号（如中国大陆手机号写“0086”）	默认四位，不足四位前面补 0	是
password	String	注册的账号密码		是
smscode	String	手机接收到的验证码		是

例子：

```
- (void)registerByPhone {
[[IOTSmartUser sharedInstance] IOTCloudSDK_registerByPhoneNum:
@"your_phone_number" zone: @"your_phone_zone" password:@"your_password"
smsCode:@"verify_code" success:^{
        NSLog(@"register success");
```

```
        } failure:^( int errNo,NSString *errMessage) {
        NSLog(@"register failure: %@", errMessage);
        }];
}
```

3.2、用户登录

使用手机号与密码进行登录。

请求参数：

参数名	类型	说明	备注	必需
phone	String	手机号码	接收验证码的手机号	是
password	String	注册的账号密码		是
zone	String	区号（如中国大陆手机号写“0086”）	默认四位，不足四位前面补 0	是

例子：

```
- (void)loginByPhoneAndPassword {
    [[IOTSmartUser sharedInstance] IOTCloudSDK_loginByPhoneNum:
@"your_phone_number" password:@"your_password"  zone: @"your_phone_zone"
success:^{
        NSLog(@"login success");
    } failure:^( int errNo,NSString *errMessage) {
        NSLog(@"login failure: %@", errMessage);
    }];
}
```

3.3、找回密码

找回密码分两步，一是发送验证码到手机，二是使用验证码和手机号重置密码。

3.3.1 发送找回密码验证码到手机

此短消息模版是青莲云找回密码消息模版，与 3.1.1 不同。

请求参数：

参数名	类型	说明	备注	必需
phone	String	手机号码	接收验证码的手机号	是
zone	String	区号（如中国大陆手机号写“0086”）	默认四位，不足四位前面补 0	是

例子：

```
- (void) sendVerifyCode {
 [[IOTSmartUser sharedInstance] IOTCloudSDK_resetPswSendVerifyCodeByPhoneNum:
@"your_phone_number" zone: @"your_phone_zone" success:^{
```

```
        NSLog(@"sendVerifyCode success");
} failure:^( int errNo,NSString *errMessage) {
        NSLog(@"sendVerifyCode failure: %@", errMessage);
}];
}
```

3.3.2、使用手机验证码重置密码

请求参数：

参数名	类型	说明	备注	必需
phone	String	手机号码	接收验证码的手机号	是
password	String	注册的账号密码		是
smsCode	String	手机接收到的验证码		是
zone	String	区号（如中国大陆手机号写“0086”）	默认四位，不足四位前面补 0	是

例子：

```
- (void)resetPassword{
    [[IOTSmartUser sharedInstance] IOTCloudSDK_resetPasswordByPhoneNum:
@"your_phone_number" password:@"your_password" smsCode:@"verify_code" zone:
@"your_phone_zone" success:^{
        NSLog(@"resetPassword success");
} failure:^( int errNo,NSString *errMessage) {
        NSLog(@"resetPassword failure: %@", errMessage);
}];
}
```

3.4、获取用户信息

登录之后的用户才能获取用户信息，如：头像、昵称、用户 id、用户邮箱等。返回 IOTSmartAccountModel 实例。

请求参数：无

例子：

```
- (void)getUserInfo{
    [[IOTSmartUser sharedInstance]
IOTCloudSDK_getAccountInfoSuccess:^( IOTSmartAccountModel *accountModel){
    NSLog(@"getUserInfo success");
} failure:^( int errNo,NSString *errMessage) {
    NSLog(@"getUserInfo failure: %@",errMessage);
    }];
}
```

3.5、获取登录状态

通过单例可直接获取用户的登录状态。

请求参数：无

例子：

```
- (void) getUserLoginState {
    BOOL isLogin= [[IOTSmartUser sharedInstance] IOTCloudSDK_getIsLogin];
}
```

3.6、退出登录

请求参数：无

例子：

```
- (void) signOut {
    [[IOTSmartUser sharedInstance] IOTCloudSDK_Signout];
}
```

4、产品管理

产品相关的所有功能对应 IOTSDK.framework 中的 IOTSmartProduct 类。

4.1、获取产品列表

通过调用该接口获取所有产品列表。返回 IOTSmartProductListModel 类对象数组。

请求参数：无

例子：

```
- (void)getProductList{
     [IOTSmartProduct IOTCloudSDK_getProductListSuccess:
^(NSArray<IOTSmartProductListModel *> *productListModelArr) {
     NSLog(@"getProductList success");
} failure:^( int errNo,NSString *errMessage) {
     NSLog(@"getProductList failure: %@", errMessage);
}];
}
```

4.2、获取产品实体类对象

获取产品实体类对象，覆盖 alloc 创建对象，以下 self.product 均表示已经创建了的产品类对象。API 请求，保证实时性可用性。

请求参数：

参数名	类型	说明	备注	必需
productId	int	产品 id		是

例子：

```
- (void)getProductObj{
   [IOTSmartProduct IOTCloudSDK_productWithProductId: @"your_productId"
success:^( IOTSmartProduct *product) {
            NSLog(@"getProduct success");
   } failure:^( int errNo,NSString *errMessage) {
            NSLog(@"getProduct failure: %@", errMessage);
   }];
}
```

4.3、获取产品上报数据点

获取产品官网上创建的可上报数据点。返回 IOTSmartProductDPModel 类对象数组。

请求参数：无

例子：

```
- (void)getProductUpDPs{
       [self.product IOTCloudSDK_getUpDPsSuccess: ^(NSArray<IOTSmartProductDPModel
*> *productDPModelArr) {
            NSLog(@"getProductUpDPs success");
} failure:^( int errNo,NSString *errMessage) {
            NSLog(@"getProductUpDPs failure: %@", errMessage);
       }];
}
```

4.4、获取产品下发数据点

获取产品官网上创建的可下发数据点。

请求参数：无

例子：

```
- (void)getProductDownDPs{
       [self.product  IOTCloudSDK_getDownDPsSuccess:
^(NSArray<IOTSmartProductDPModel *> *productDPModelArr) {
            NSLog(@"getProductDownDPs success");
} failure:^( int errNo,NSString *errMessage) {
            NSLog(@"getProductDownDPs failure: %@", errMessage);
      }];
}
```

5、设备管理

设备相关的所有功能对应 IOTSDK.framework 中的 IOTSmartDevice 类。

5.1、获取绑定设备列表

通过调用该接口获取所有设备列表。返回 IOTSmartDevice 类对象数组。

请求参数：无

例子：

```
- (void)getDeviceList{
    [IOTSmartDevice IOTCloudSDK_getDeviceListSuccess: ^(NSArray<IOTSmartDevice
*> * deviceLisArr) {
        NSLog(@"getDeviceList success");
} failure:^( int errNo,NSString *errMessage) {
        NSLog(@"getDeviceList failure: %@", errMessage);
    }];
}
```

5.2、获取设备详细信息

获取指定设备的详细信息。请求成功返回 IOTSmartDeviceInfoModel 实体类对象。

请求参数：无

例子：

```
- (void)getDeviceInfo{
    [self.device IOTCloudSDK_getDeviceInfoSuccess: ^(IOTSmartDeviceInfoModel *
deviceModel){
        NSLog(@"getDeviceInfo success");
} failure:^( int errNo,NSString *errMessage) {
        NSLog(@"getDeviceInfo failure: %@", errMessage);
    }];
}
```

5.3、修改设备名称

手机修改已经绑定的某设备的显示名称。

请求参数：

参数名	类型	说明	备注	必需
deviceName	String	设备显示名	修改后的显示名	是

例子：

```
- (void)editDeviceName{
    [self.device IOTCloudSDK_modifyDeviceNameByName: @"your_Device_Name"
success : ^{
        NSLog(@"editDeviceName success");
} failure:^( int errNo,NSString *errMessage) {
        NSLog(@"editDeviceName failure: %@", errMessage);
    }];
```

```
}
```

5.4、检查用户绑定设备是否可升级

获取当前登录用户所绑定的设备中能够进行 OTA 升级的设备列表。

请求参数：无

例子：

```
- (void)checkUserDeviceOTA{
    [IOTSmartDevice IOTCloudSDK_checkUserDeviceOTASuccess:
^(NSArray<IOTSmartDeviceOTAStatusListModel *> *OTAStatusListModelArr) {
    NSLog(@"check success");
  } failure:^(int errNo, NSString *errMessage) {
    NSLog(@"check failure: %@", errMessage);
  }];
}
```

5.5、检查用户绑定设备的 OTA 升级状态

给蓝牙设备传输 OTA 升级包完成，等待设备重启后调用此方法可查询 OTA 升级结果。

请求参数：

参数名	类型	说明	备注	必需
versionNum	String	检查的版本号	无	是

例子：

```
- (void)checkOTAState{
    [self.device IOTCloudSDK_checkDeviceOTADownloadStatusWithVersionNum:
@"versionNum" success:^(IOTSmartDeviceOTADownloadProgressModel
*OTADownloadProgressModel)  {
    NSLog(@"check success");
  } failure:^(int errNo, NSString *errMessage) {
    NSLog(@"check failure: %@", errMessage);
  }];
}
```

6、蓝牙 Mesh

蓝牙 Mesh 相关的功能对应 IOTSmartBlueToothMesh 类（单例）。

6.1、创建 Mesh 网络

使用此方法可创建并初始化一个 Mesh 网络。

请求参数：

参数名	类型	说明	备注	必需
name	String	网络的名称	无	是
ttl	int	消息的最大转发次数	Time To Live 的缩写	是

例子：

```
- (void)createMesh {
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_createBLEMeshWithMeshName:@"name" meshTTL:@"ttl"
success:^(IOTSmartMeshModel *mesh) {
        NSLog(@"createMesh success");
    } failure:^(NSString *errMsg) {
        NSLog(@"createMesh failure: %@", errMsg);
    }];
}
```

6.2、删除 Mesh 网络

使用此方法传入一个 Mesh 网络的对象，可删除该网络并重置网络下的所有节点。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要删除的网络对象	无	是

例子：

```
- (void)removeMesh {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_removeBLEMesh:@"mesh"
success:^ {
        NSLog(@"remove success");
    } failure:^(NSString *errMsg) {
        NSLog(@"remove failure: %@", errMsg);
    }];
}
```

6.3、获取所有 Mesh 网络

使用此方法可获取当前手机保存的所有 Mesh 网络的网络对象。

请求参数：无。

例子：

```
- (void)getAllMesh {
```

```
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_getAllBLEMeshSuccess:^(NSArray<IOTSmartMeshModel *> *meshArr) {
        NSLog(@"getAllMesh success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getAllMesh failure: %@", errMsg);
    }];
}
```

6.4、连接 Mesh 网络

使用此方法可连接到 Mesh 网络的某个代理节点上，进而可进行 App 与 Mesh 中节点的交互，使用前提是该 Mesh 网络中有代理节点。

请求参数：

参数名	类型	说明	备注	必需
timeout	int	连接的超时时间	无	是
mesh	IOTSmartMeshModel	要连接的 mesh 网络	无	是

例子：

```
- (void)connectMesh {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_connectBLEMesh:@"mesh"
timeout:@"timeout" success:^ {
        NSLog(@"connect success");
    } failure:^(NSString *errMsg) {
        NSLog(@"connect failure: %@", errMsg);
    }];
}
```

6.5、断开 Mesh 网络

使用此方法可断开 App 与 Mesh 网络的连接。若已断开则直接返回成功。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要断开的 mesh 网络	无	是

例子：

```
- (void)disconnectMesh {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_disConnectBLEMesh:@"mesh"
success:^ {
        NSLog(@"disconnect success");
    } failure:^(NSString *errMsg) {
        NSLog(@"disconnect failure: %@", errMsg);
```

```
    }];
}
```

6.6、获取 Mesh 网络的所有节点

使用此方法可获取到某个 Mesh 网络下的所有节点。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要获取节点的 mesh 网络	无	是

例子：

```
- (void)getAllNodes {
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_getAllNodeWithBLEMesh:@"mesh" success:^(NSArray<IOTSmartNode *>
*nodeArr) {
        NSLog(@"getAllNodes success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getAllNodes failure: %@", errMsg);
    }];
}
```

6.7、修改网络的名称及 TTL

使用此方法可修改网络的名称及 Mesh 网络的 TTL。前提要连接该 Mesh 网络。

请求参数：

参数名	类型	说明	备注	必需
meshName	String	修改后的网络名称	无	否
meshTTL	int	修改后的网络 TTL	无	否
mesh	IOTSmartMeshModel	要修改的 mesh 网络	无	是

例子：

```
- (void)modifyMeshNameAndTTL {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_modifyMeshName:
@"meshName" meshTTL:@"meshTTL" withMesh:@"mesh" success:^{
        NSLog(@"modify success");
    } failure:^(NSString *errMsg) {
        NSLog(@"modify failure: %@", errMsg);
    }];
}
```

6.8、搜索周围的可添加节点

使用此方法可搜索周围没有被添加到 Mesh 网络的节点。

请求参数：

参数名	类型	说明	备注	必需
timeout	int	搜索的超时时间	无	是
mesh	IOTSmartMeshModel	要搜索节点的 mesh 网络	无	是

例子：

```
- (void)searchLeisureNode {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_searchNodeWithTimeout:
@"timeout" withMesh:@"mesh" success:^(NSString *nodeName, NSString *mac) {
        NSLog(@"search success");
    } failure:^(NSString *errMsg) {
        NSLog(@"search failure: %@", errMsg);
    }];
}
```

6.9、Mesh 网络添加节点

使用此方法可以将搜索到的没有被添加到 Mesh 网络的节点添加到该 Mesh 网络。前提条件要先进行搜索。

请求参数：

参数名	类型	说明	备注	必需
mac	String	节点的 mac 地址	无	是
productId	int	产品 ID	无	是
productKey	String	产品的 Key	无	是
mesh	IOTSmartMeshModel	要添加节点的 mesh 网络	无	是

例子：

```
- (void)meshAddNode {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_addNodeWithMac:@"mac"
productId: @"productId" productKey:@"productKey" toMesh:@"mesh"
success:^(IOTSmartNode *node) {
        NSLog(@"addNode success");
    } failure:^(NSString *errMsg) {
        NSLog(@"addNode failure: %@", errMsg);
    }];
}
```

6.10、重置节点

使用此方法可将节点重置为未加入 Mesh 网络的节点，并将节点信息从传入的 mesh 网络中删除。注意与删除节点信息进行区分。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
node	IOTSmartNode	需要重置的节点	无	是
mesh	IOTSmartMeshModel	要重置节点的 mesh 网络	无	是

例子：

```
- (void)resetNode {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_resetNodeWithNode:@"node" fromMesh:@"mesh" success:^ {
        NSLog(@"resetNode success");
    } failure:^(NSString *errMsg) {
        NSLog(@"resetNode failure: %@", errMsg);
    }];
}
```

6.11、删除节点信息

删除本地保存的节点信息，将节点从网络的节点列表中删除，不管节点是否被重置。

请求参数：

参数名	类型	说明	备注	必需
node	IOTSmartNode	需要删除信息的节点	无	是
mesh	IOTSmartMeshModel	要删除节点的 mesh 网络	无	是

例子：

```
- (void)deleteNodeInfo {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_deleteNodeWithNode:@"node" fromMesh:@"mesh"];
}
```

6.12、修改节点名称

使用此方法可修改网络内指定 mac 对应节点的名称。

请求参数：

参数名	类型	说明	备注	必需
nodeName	String	修改后的节点的名称	无	是
mac	String	要修改名称的节点 mac	无	是
mesh	IOTSmartMeshModel	要修改节点名称的 mesh 网络	无	是

例子：

```
- (void)modifyNodeName {
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_modifyNodeName:@"nodeName" nodeMac:@"mac" fromMesh:@"mesh"
success:^ {
        NSLog(@"modify success");
    } failure:^(NSString *errMsg) {
        NSLog(@"modify failure: %@", errMsg);
    }];
}
```

6.13、向节点发送二进制透传消息

使用此方法可向网络内的任意节点发送二进制透传消息。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
data	NSData	透传的数据	无	是
mac	String	目标节点的 mac	无	是
mesh	IOTSmartMeshModel	发送透传消息的 mesh 网络	无	是

例子：

```
- (void)pushMessage {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_pushData:@"data" mac:@"mac"
withMesh:@"mesh" success:^ {
        NSLog(@"push success");
    } failure:^(NSString *errMsg) {
        NSLog(@"push failure: %@", errMsg);
    }];
}
```

6.14、向节点发送数据点消息

使用此方法可向网络内的任意节点下发数据点消息。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
dpArr	NSArray	数组中存放数据点对象	无	是
mac	String	目标节点的 mac	无	是
mesh	IOTSmartMeshModel	发送数据点的 mesh 网络	无	是

例子：

```
- (void)publishDps {
```

```
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_publishDPs:@"dpArr"
mac:@"mac" withMesh:@"mesh" success:^ {
        NSLog(@"publish success");
    } failure:^(NSString *errMsg) {
        NSLog(@"publish failure: %@", errMsg);
    }];
}
```

6.15、获取节点的订阅节点列表

使用此方法可获取某节点所订阅的所有节点。如：A 订阅了 B、C、D，参数中传 A 的 mac，返回 B、C、D 的对象数组。

请求参数：

参数名	类型	说明	备注	必需
mac	String	节点的 mac	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)getSubscribeArr {
    [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_ getSubscribeNodeListWithMac:
@"mac" withMesh:@"mesh" success:^(NSArray<IOTSmartNode *> *nodeArr) {
        NSLog(@"getSubscribeArr success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getSubscribeArr failure: %@", errMsg);
    }];
}
```

6.16、让节点去订阅另一个节点

此方法可让节点 1（源节点）去订阅节点 2（被订阅节点）。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
srcMac	String	源节点	无	是
subMac	String	源节点需要订阅的节点	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
```

```
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_addSubscribeWithSourceNodeMac:@"srcMac"
subscribeNodeMac:@"subMac" withMesh:@"mesh" success:^ {
        NSLog(@"subscribe success");
    } failure:^(NSString *errMsg) {
        NSLog(@"subscribe failure: %@", errMsg);
    }];
}
```

6.17、让节点取消订阅某个节点

使用此方法可让节点 1（源节点）取消订阅节点 2（被订阅节点），前提是节点 1 已经订阅了节点 2。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
srcMac	String	源节点	无	是
subMac	String	源节点需要取消订阅的节点	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
    [[IOTSmartBlueToothMesh shareInstance]
IOTCloudSDK_removeSubscribeWithSourceNodeMac:@"srcMac"
subscribeNodeMac:@"subMac" withMesh:@"mesh" success:^ {
        NSLog(@"remove subscribe success");
    } failure:^(NSString *errMsg) {
        NSLog(@"remove subscribe failure: %@", errMsg);
    }];
}
```

6.18、下发极速小数据点

使用此方法可向某节点极速下发数据点消息。注意：此方法一次只能发送一个数据点且数据点只能使用（布尔、整形、枚举、浮点）类型。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
dpModel	IOTSmartBlueToothDPModel	数据点对象	无	是
mac	String	要发送到的节点的 mac	无	是

mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
   [[IOTSmartBlueToothMesh shareInstance] IOTCloudSDK_topSpeedPublishSmallDP:
@"dpModel" mac:@"mac" withMesh:@"mesh"];
}
```

6.19、实时接收节点发送二进制透传

用户遵循 IOTSmartBlueToothMeshDelegate 并实现 IOTCloudSDK_didReceivedPushData:withMac:meshUUID:方法即可实时的接收到网络内的节点发送的二进制透传消息。

6.20、实时接收节点上报数据点

用户遵循 IOTSmartBlueToothMeshDelegate 并实现 IOTCloudSDK_didReceivedDPArr:withMac:meshUUID:方法即可实时的接收到网络内的节点上报数据点消息。前提需要先连接 Mesh 网络。

6.21、实时接收节点上报极速小数据点

用户需要遵循 IOTSmartBlueToothMeshDelegate 协议并实现其中的代理方法 IOTCloudSDK_didReceivedTopSpeedSmallDP:withMac:meshUUID:即可实时的接收到网络内节点上报的极速小数据点。前提需要连接 Mesh 网络。

6.22、监听网络连接和断开

手机端要监听 Mesh 网络的连接或非正常断开（主动断开只执行回调）。需将自身作为 IOTSmartBlueToothMeshConnectionStatus 通知的观察者。当手机与 Mesh 网络连接成功或非正常断开连接时即可监听到对应消息。

7、云端蓝牙 Mesh

云端蓝牙 Mesh 对应的所有功能都在 IOTSmartCloudBlueToothMesh（单例）类中。本节全部功能的前提条件：项目中必须要导入 IOTSDK 且必须要先进行登录。

7.1、创建 Mesh 网络

使用此方法可创建并初始化一个 Mesh 网络，并将此 Mesh 网络的信息保存至云端与当前登录用户进行关联。

请求参数：

参数名	类型	说明	备注	必需
name	String	网络的名称	无	是
ttl	int	消息的最大转发次数	Time To Live 的缩写	是

例子：

```
- (void)createMesh {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_createBLEMeshWithMeshName:@"name" meshTTL:@"ttl"
success:^(IOTSmartMeshModel *mesh) {
        NSLog(@"createMesh success");
    } failure:^(NSString *errMsg) {
        NSLog(@"createMesh failure: %@", errMsg);
    }];
}
```

7.2、删除指定的 Mesh 网络

使用此方法传入一个 Mesh 网络的对象，可删除该网络并重置网络下的所有节点。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要删除的网络对象	无	是

例子：

```
- (void)removeMesh {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_removeBLEMesh:@"mesh" success:^ {
        NSLog(@"remove success");
    } failure:^(NSString *errMsg) {
        NSLog(@"remove failure: %@", errMsg);
    }];
}
```

7.3、获取所有 Mesh 网络

使用此方法可获取当前登录的用户拥有的所有 Mesh 网络的对象。

请求参数：无。

例子：

```
- (void)getAllMesh {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_getAllBLEMeshSuccess:^(NSArray<IOTSmartMeshModel *> *meshArr) {
        NSLog(@"getAllMesh success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getAllMesh failure: %@", errMsg);
    }];
}
```

7.4、连接 Mesh 网络

使用此方法可连接到指定 Mesh 网络，进而可进行 App 与 Mesh 中节点的数据交互。

请求参数：

参数名	类型	说明	备注	必需
timeout	int	连接的超时时间	无	是
mesh	IOTSmartMeshModel	要连接的 mesh 网络	无	是

例子：

```
- (void)connectMesh {
    [[IOTSmartCloudBlueToothMesh shareInstance]
    IOTCloudSDK_connectBLEMesh:@"mesh" timeout:@"timeout" success:^ {
        NSLog(@"connect success");
    } failure:^(NSString *errMsg) {
        NSLog(@"connect failure: %@", errMsg);
    }];
}
```

7.5、断开 Mesh 网络

使用此方法可断开 App 与 Mesh 网络的连接。若已断开则直接返回成功。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要断开的 mesh 网络	无	是

例子：

```
- (void)disconnectMesh {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_disConnectBLEMesh:@"mesh" success:^ {
        NSLog(@"disconnect success");
    } failure:^(NSString *errMsg) {
        NSLog(@"disconnect failure: %@", errMsg);
    }];
}
```

7.6、获取 Mesh 网络的所有节点

使用此方法可从云端获取指定 Mesh 网络下的所有节点。

请求参数：

参数名	类型	说明	备注	必需
mesh	IOTSmartMeshModel	要获取节点的 mesh 网络	无	是

例子：

```
- (void)getAllNodes {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_getAllNodeWithBLEMesh:@"mesh" success:^(NSArray<IOTSmartNode *>
*nodeArr) {
        NSLog(@"getAllNodes success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getAllNodes failure: %@", errMsg);
    }];
}
```

7.7、修改网络的名称及 TTL

使用此方法可修改 Mesh 网络的名称及 Mesh 网络的 TTL。前提要连接该 Mesh 网络。

请求参数：

参数名	类型	说明	备注	必需
meshName	String	修改后的网络名称	无	否
meshTTL	int	修改后的网络 TTL	无	否
mesh	IOTSmartMeshModel	要修改的 mesh 网络	无	是

例子：

```
- (void)modifyMeshNameAndTTL {
    [[IOTSmartCloudBlueToothMesh shareInstance] IOTCloudSDK_modifyMeshName:
@"meshName" meshTTL:@"meshTTL" withMesh:@"mesh" success:^{
        NSLog(@"modify success");
    } failure:^(NSString *errMsg) {
        NSLog(@"modify failure: %@", errMsg);
    }];
}
```

7.8、搜索周围的可入网节点

使用此方法可搜索周围没有被添加到 Mesh 网络的节点。

请求参数：

参数名	类型	说明	备注	必需
timeout	int	搜索的超时时间	无	是
mesh	IOTSmartMeshModel	要搜索节点的 mesh 网络	无	是

例子：

```
- (void)searchLeisureNode {
```

```
    [[IOTSmartCloudBlueToothMesh shareInstance] IOTCloudSDK_searchNodeWithTimeout:
@"timeout" withMesh:@"mesh" success:^(NSString *nodeName, NSString *mac) {
        NSLog(@"search success");
    } failure:^(NSString *errMsg) {
        NSLog(@"search failure: %@", errMsg);
    }];
}
```

7.9、Mesh 网络添加节点

使用此方法可将搜索到的没有被添加到 Mesh 网络的节点添加到该 Mesh 网络。前提条件要先进行搜索。

请求参数：

参数名	类型	说明	备注	必需
mac	String	节点的 mac 地址	无	是
productId	int	产品 ID	无	是
productKey	String	产品的 Key	无	是
mesh	IOTSmartMeshModel	要添加节点的 mesh 网络	无	是

例子：

```
- (void)meshAddNode {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_addNodeWithMac:@"mac" productId: @"productId"
productKey:@"productKey" toMesh:@"mesh" success:^(IOTSmartNode *node) {
        NSLog(@"addNode success");
    } failure:^(NSString *errMsg) {
        NSLog(@"addNode failure: %@", errMsg);
    }];
}
```

7.10、重置节点

使用此方法可将节点重置为未加入 Mesh 网络的节点，并将节点信息从指定的 mesh 网络中删除。注意与删除节点信息进行区分。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
node	IOTSmartNode	需要重置的节点	无	是
mesh	IOTSmartMeshModel	要重置节点的 mesh 网络	无	是

例子：

```
- (void)resetNode {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_resetNodeWithNode:@"node" fromMesh:@"mesh" success:^ {
        NSLog(@"resetNode success");
    } failure:^(NSString *errMsg) {
        NSLog(@"resetNode failure: %@", errMsg);
    }];
}
```

7.11、删除节点信息

此方法用来清除由于某些原因导致节点已不属于 Mesh 网络但节点信息还存在的节点。其他情况下请勿调用此方法，可能会导致节点信息从云端和本地清除了但节点还在 Mesh 网络内的情况，此时就必须用户手动在设备上进行重置。若要将某节点从网络中移除请使用重置节点方法。

请求参数：

参数名	类型	说明	备注	必需
node	IOTSmartNode	需要删除信息的节点	无	是
mesh	IOTSmartMeshModel	要删除节点的 mesh 网络	无	是

例子：

```
- (void)deleteNodeInfo {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_deleteNodeWithNode:@"node" fromMesh:@"mesh" success:^ {
        NSLog(@"deleteNode success");
    } failure:^(NSString *errMsg) {
        NSLog(@"deleteNode failure: %@", errMsg);
    }];
}
```

7.12、修改节点名称

使用此方法可修改网络内指定 mac 对应节点的名称。

请求参数：

参数名	类型	说明	备注	必需
nodeName	String	修改后的节点的名称	无	是
mac	String	要修改名称的节点 mac	无	是
mesh	IOTSmartMeshModel	要修改节点名称的 mesh 网络	无	是

例子：

```
- (void)modifyNodeName {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_modifyNodeName:@"nodeName" nodeMac:@"mac" fromMesh:@"mesh"
success:^ {
        NSLog(@"modify success");
    } failure:^(NSString *errMsg) {
        NSLog(@"modify failure: %@", errMsg);
    }];
}
```

7.13、向节点发送二进制透传消息

使用此方法向指定的 Mesh 网络内的指定节点发送二进制透传消息。前提条件：需要连接网络。

请求参数：

参数名	类型	说明	备注	必需
data	NSData	透传的数据	无	是
mac	String	目标节点的 mac	无	是
mesh	IOTSmartMeshModel	发送透传消息的 mesh 网络	无	是

例子：

```
- (void)pushMessage {
    [[IOTSmartCloudBlueToothMesh shareInstance] IOTCloudSDK_pushData:@"data"
mac:@"mac" withMesh:@"mesh" success:^ {
        NSLog(@"push success");
    } failure:^(NSString *errMsg) {
        NSLog(@"push failure: %@", errMsg);
    }];
}
```

7.14、向节点发送数据点消息

使用此方法可向指定 Mesh 网络的指定节点下发数据点消息。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
dpArr	NSArray	数组中存放数据点对象	无	是
mac	String	目标节点的 mac	无	是
mesh	IOTSmartMeshModel	发送数据点的 mesh 网络	无	是

例子：

```
- (void)publishDps {
```

```
    [[IOTSmartCloudBlueToothMesh shareInstance] IOTCloudSDK_publishDPs:@"dpArr"
mac:@"mac" withMesh:@"mesh" success:^ {
        NSLog(@"publish success");
    } failure:^(NSString *errMsg) {
        NSLog(@"publish failure: %@", errMsg);
    }];
}
```

7.15、获取节点的订阅节点列表

使用此方法可获取某节点所订阅的所有节点。如：A 订阅了 B、C、D，参数中传 A 的 mac，返回 B、C、D 的对象数组。

请求参数：

参数名	类型	说明	备注	必需
mac	String	节点的 mac	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)getSubscribeArr {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_getSubscribeNodeListWithMac:@"mac" withMesh:@"mesh"
success:^(NSArray<IOTSmartNode *> *nodeArr) {
        NSLog(@"getSubscribeArr success");
    } failure:^(NSString *errMsg) {
        NSLog(@"getSubscribeArr failure: %@", errMsg);
    }];
}
```

7.16、让节点去订阅另一个节点

此方法可让节点 1（源节点）去订阅节点 2（被订阅节点）。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
srcMac	String	源节点	无	是
subMac	String	源节点需要订阅的节点	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
```

```
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_addSubscribeWithSourceNodeMac:@"srcMac"
subscribeNodeMac:@"subMac" withMesh:@"mesh" success:^ {
        NSLog(@"subscribe success");
    } failure:^(NSString *errMsg) {
        NSLog(@"subscribe failure: %@", errMsg);
    }];
}
```

7.17、让节点取消订阅某个节点

使用此方法可让节点 1（源节点）取消订阅节点 2（被订阅节点），前提是节点 1 已经订阅了节点 2。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
srcMac	String	源节点	无	是
subMac	String	源节点需要取消订阅的节点	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
    [[IOTSmartCloudBlueToothMesh shareInstance]
IOTCloudSDK_removeSubscribeWithSourceNodeMac:@"srcMac"
subscribeNodeMac:@"subMac" withMesh:@"mesh" success:^ {
        NSLog(@"remove subscribe success");
    } failure:^(NSString *errMsg) {
        NSLog(@"remove subscribe failure: %@", errMsg);
    }];
}
```

7.18、向节点下发极速小数据点

使用此方法可向某节点极速下发数据点消息。注意：此方法一次只能发送一个数据点且数据点只能使用（布尔、整形、枚举、浮点）类型。前提条件：需要连接网络

请求参数：

参数名	类型	说明	备注	必需
dpModel	IOTSmartBlueToothDPModel	数据点对象	无	是
mac	String	要发送到的节点的 mac	无	是
mesh	IOTSmartMeshModel	节点所属的 mesh 网络	无	是

例子：

```
- (void)addSubscribeNode {
   [[IOTSmartCloudBlueToothMesh shareInstance] IOTCloudSDK_topSpeedPublishSmallDP:@"dpModel" mac:@"mac" withMesh:@"mesh"];
}
```

7.19、实时接收节点发送二进制透传

用户遵循 IOTSmartCloudBlueToothMeshDelegate 代理协议并实现协议中的 IOTCloudSDK_didReceivedPushData:withMac:meshUUID:方法可实时的接收到网络内的节点发送的二进制透传消息。

7.20、实时接收节点上报数据点

遵循 IOTSmartCloudBlueToothMeshDelegate 并实现 IOTCloudSDK_didReceivedDPArr:withMac:meshUUID:方法可实时的接收到网络内的节点上报数据点消息。前提需要先连接 Mesh 网络。

7.21、实时接收节点上报极速小数据点消息

用户遵循 IOTSmartCloudBlueToothMeshDelegate 协议并实现其中的 IOTCloudSDK_didReceivedTopSpeedSmallDP:withMac:meshUUID: 方法可实时的接收到网络内节点上报的极速小数据点。前提需要连接 Mesh 网络。

7.22、监听网络连接和断开

手机端要监听 Mesh 网络的连接或非正常断开（主动断开只执行回调）。需将自身作为 IOTSmartCloudBlueToothMeshConnectionStatus 通知的观察者。当手机与 Mesh 网络连接成功或非正常断开连接时即可监听到对应消息。