**ZYIOT Account SDK**

**使用说明**

版本：V2.4.0 日期：2019-06-27

2019

**专注云技术、共筑云安全**

**关于本文档**

ZYIOT Account SDK的使用说明。

**发布说明**

|  |  |  |
| --- | --- | --- |
| **日期** | **版本** | **发布说明** |
| 2019-06-27 | V2.4.0 | 1. 增加Account用户相关功能、分享功能、转移设备功能、以及套餐功能  2. 增加HttpDNS功能  3. 增加消息中心接口 |
| 2018-12-19 | V2.3.0 | 接口优化，增加SDK Login状态监听回调接口。 |
| 2018-12-03 | V2.2.1 | 1. 新增网关子设备功能 2. 新增Open Acount支持 |
| 2017-06-22 | V1.0.0 | 首次发布 |

**文档变更通知**

用户可咨询技术支持人员或通过邮箱：[info@zhuyun-it.com](mailto:info@zhuyun-it.com) 获取最新文档。

**目录**

[1 接入必读 3](#_Toc4132)

[2 简介 4](#_Toc6204)

[3 名词解释 5](#_Toc31913)

[4 功能介绍 7](#_Toc563)

[5 安装SDK 9](#_Toc7766)

[6 网络配置http协议可用 11](#_Toc13337)

[7 主要流程 12](#_Toc8956)

[7.1 初始化SDK 12](#_Toc14784)

[7.2 手机账号登录 12](#_Toc12130)

[7.3 添加设备、分享设备、编辑被分享权限、删除设备 12](#_Toc13345)

[7.4 获取用户下的设备列表 15](#_Toc31393)

[7.5 获取指定设备的所有属性 16](#_Toc22485)

[7.6 控制设备 16](#_Toc24911)

[7.7 设备固件升级 19](#_Toc2060)

[7.8 关闭IOT Client 21](#_Toc5891)

[7.9 获取服务器HttpDNS信息 21](#_Toc14982)

# 接入必读

IOS的demo已经包含了开放平台SDK的静态库和头文件，请将SDK库文件和头文件拷贝到项目工程根目录下的ZYIOTSDK文件夹中。

# 简介

本文档用于说明筑云开放平台SDK IOS版本接口之间的关系以及接口调用顺序，对开放平台SDK IOS版本主要流程都有详细说明和代码示例。主要有功能介绍、安装说明、权限配置和主要流程介绍。

# 名词解释

|  |  |
| --- | --- |
| 名词 | 注解 |
| tenantId | 筑云提供的一个标识ID |
| keyhash | 设备唯一标识（序列号） |
| devTypeId | 设备类型ID |
| DNSKey | 使用筑云HttpDNS功能的APP，请提前到筑云申请一个DNSKey。用于获取APP对应的服务器DNS信息。 |
| DNS（HttpDNS） | DNS服务器，存储了不同APP对应服务器的DNS地址信息、支持的Tenant账号等。 |
| ZOT | 用户与设备相关功能的业务处理模块（处理除FOTA与IOT之外的业务）。SDK接口分为三大模块，分别是ZOT+DNS、IOT、FOTA。 |
| IOT Client | SDK中专门用来与设备进行通信的客户端 |
| Event | IOT Client与设备进行通信的消息封装（比如包含属性名、属性值），详见API文档。 |
| FOTA | 设备固件升级模块 |
| 设备属性 | 根据设备类型由厂家规定，包含属性名、属性值类型、属性值。（属性规定，详见厂家设备属性定义文档）。 |
| 属性值类型 | 表示Event封装的属性值的类型，1是string；2是int；4是hex。 |
| hexTypeSize | 通过Event发送语音文件时，分片发送时单片内容的最大数据长度。 |
| endpointType | 端点类型，分为：0：wifi; 1：app; 2：app下挂设备(app直连子设备); 3：网关下挂设备 |
| parentKeyhash | 父设备keyhash，针对app下挂设备和网关下挂设备，为这两种类型向上一级的设备keyhash |
| authority | 用户对于设备的权限（管理员无此字段），JSON格式内容。 |
| ZY Account | ZYIOT Account，使用手机或邮箱注册的筑云账号 |
| Open Account | ZYIOT Open Account，接入筑云的第三方平台的用户。（Open Account 接入SDK的使用说明见另一份文档《ZYIOT\_Open\_Account SDK使用说明\_ios.docx》） |

表 3-1

# 功能介绍

|  |  |
| --- | --- |
| 功能 | 说明 |
| 用户管理 | 支持手机账号登录，支持查看设备下的用户列表，查看用户下的设备列表 |
| 临时用户管理 | 支持将设备分享给临时用户，可增加临时用户、修改临时用户权限、删除临时用户（临时用户使用设备可以不需要自己的账号，需要被分享时得到的tempUserId） |
| 设备管理 | 支持绑定设备、删除设备、控制设备，还可将设备授权分享给其他用户、可编辑被分享用户的权限、可删除被分享用户、转移设备、为设备绑定摄像头信息 |
| 设备记录 | 获取设备历史记录（设备异常上下线、操控记录，以及授权分享记录） |
| 联动与定时 | 支持设备与设备的联动；支持设备定时执行动作 |
| 触发器 | 支持为自己的设备增加自定义触发器、编辑和删除自定义触发器（减少触发情况，达到更精确触发，得到需要的消息） |
| 套餐管理 | 可以获取设备/短信套餐信息，提交购买设备/短信套餐成功的结果 |
| 消息中心管理 | 支持获取消息类型列表（如厂家资讯、服务器更新等）、支持获取消息列表 |
| HttpDNS | 支持获取APP对应的可用服务器DNS列表信息 |

表4-1

# 安装SDK

SDK安装方式：使用完整SDK包安装。

1. 解压缩ZYIOTSDK.zip；
2. 创建一个新的XCode工程：

a. 导入SDK库libZYIOTSDK.a和include文件夹内的所有头文件；

b. 导入libcryptopp.a依赖库，如图5-1；

c. 添加libc++.tbd系统库，如图5-2。

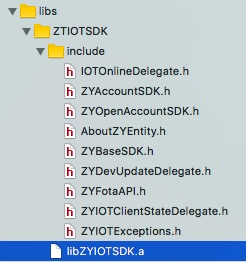
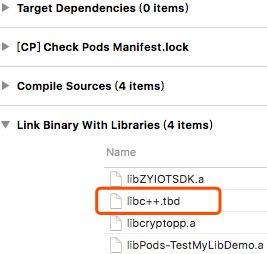
 

图5-1添加依赖库 图5-2 添加系统库

1. 使用cocoapods的方式导入以下3个依赖库：

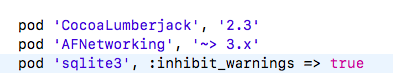


图5-3

1. 添加Other Linker Flags为-ObjC。注意区分大小写。

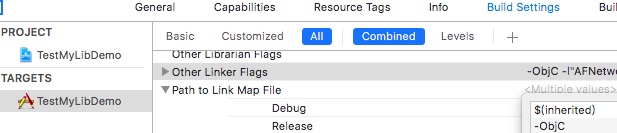


图5-4

1. 关闭目标target的bitcode功能 Build Settings->Enable Bitcode设置为No。

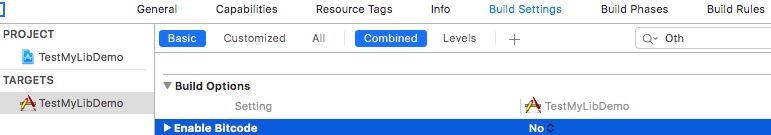


图5-5

1. 配置完成。

# 网络配置http协议可用

网络配置是在工程的info.plist文件中进行配置。在Xcode工程的文件导航栏中找到该文件，增加栏目：App Transport Security Setting为dictionary类型，并增加其下键值对：Allow Arbitary Loads=YES。

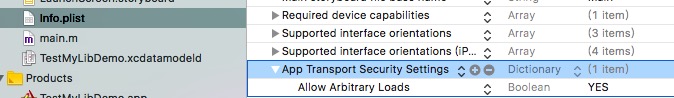


图6-1

# 主要流程

## 初始化SDK

self.zySdk=[ZYAccountSDK getZYAccountSDKInstance];//获取ZYSDK实例

[self.zySdk setTenantId:2];//指定SDK使用的tenantId

[self.zySdk setZotDomain:@"zhuyun.f3322.net"];//ZOT模块服务器IP入口

[self.zySdk setZotPort: 9390];

[self.zySdk setIOTServer:@"zhuyun.f3322.net"];//IOT模块服务器多IP入口，多个时用英文分号隔开

[[ZYFotaAPI getFotaInstance] setFotaDomain:@"zhuyun.f3322.net"];//设置Fota服务器地址

[[ZYFotaAPI getFotaInstance] setFotaPort:9324];//设置Fota端口号

//设置SDK的登录状态监听（当用户登录状态切换时调用，例如登录状态过期、异地登录等）

[self.zySdk setSDKLoginDelegate:self];

## 手机账号登录

登录后才可以正常使用SDK内API，手机账号登录后SDK将保存登录状态，直到15天未调用API登录状态过期或者账号在其他手机登录。

*//使用手机登录login(account, password, phonePush, channelId, language,listenentCallback),  
// 参数说明：account和password是必须的，phonePush、channelId、language都是与推送功能相关，如无需推送功能可不传*[self.zySdk loginWithAccount:account password:pwd phonePush:PhonePushType\_development channelId:nil language:nil completion:^(ZYUserToken \*token, int retcode, NSString \*errDescription) {

NSString \*log=[NSString stringWithFormat:@"%d login结果=%@,e=%@,result=%@",retcode,token,errDescription,retcode==200?@"登录成功，SDK启动成功":@"SDK启动失败"];

[weakSelf updateLogsText:log];

//若登录失败，则后续SDK接口无法正常使用

}];

## 添加设备、分享设备、编辑被分享权限、删除设备

支持添加绑定设备、删除设备、分享设备、编辑设备用户权限等。

代码示例：

添加设备：

//绑定设备（扫描设备二维码得到keyhash ,devTypeId），设备只允许一个用户绑定，绑定后用户是设备的管理员，level=1是一级用户。可以分享设备给其他用户，其他用户为二级用户，level=2.

[self.zySdk bindDevWithKeyhash:targetKeyhash devTypeId:self.devTypeId completion:^(int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"绑定设备成功");

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"bindDev[%@:%@].retcode=%d,errText=%@",weakSelf.devTypeId,targetKeyhash,retcode,errDescription]];

}];

删除设备：

//用户删除设备，根据指定targetKeyhash设备，可以删除设备，删除后将无法控制设备。一级用户（管理员）删除设备将连带删除设备下的所有用户(包括二级用户和临时用户),二级用户删除设备则只是自己失去设备控制权限。

[self.zySdk deleteDeviceWithKeyhash:targetKeyhash completion:^(int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"用户删除设备成功");

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"deleteDev .retcode=%d,errText=%@",retcode,errDescription]];

}];

分享设备、编辑分享的用户权限，以及删除被分享的用户：

long long now=((long long)[[NSDate date] timeIntervalSince1970]);

NSString \*authUserId=@"5b20de5c8bec6dcffc8315cb";//8账号的

//设备管理员用户将设备授权分享给其他用户（此时被分享的用户就是指定设备的二级用户），指定设备targetKeyhash然后将设备分享给指定的用户（已注册筑云的手机或邮箱），authority为空时表示全时访问即每一天每时刻都可以控制设备（具体其他权限详见API ZYAccountSDK.h接口文档），remark是备注内容可以是昵称。

[self.zySdk authorizeUserWithKeyhash:targetKeyhash account:@"12345678888" authority:[AuthorityEntity initAuthorityWithValidType:0] remark:@"妹妹" completion:^(ZYAuthUser \*authUser,int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"2 设备授权分享给用户成功 =%@",authUser);

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"authUsertoDev2 .retcode=%d,errText=%@",retcode,errDescription]];

}];

AuthorityEntity2 \*authority2=[[AuthorityEntity2 alloc] init];

//创建权限封装:每周一周二的一点到十点允许用户控制设备

authority2.time1=60;//1:00

authority2.time2=36000;//10:00

authority2.weekMode=@"12";//周一、周二

authority2.shareTime=now;

//设备管理员用户编辑二级用户权限（此时被分享的用户就是指定设备的二级用户），指定设备targetKeyhash，然后修改设备分享给指定的用户authUserId（用户ID）的权限，authority为空时表示全时访问即每一天每时刻都可以控制设备（具体其他权限详见API ZYAccountSDK.h接口文档），remark是备注内容可以是昵称。

[self.zySdk modifyAuthUserWithKeyhash:targetKeyhash authUserId:authUserId authority:authority2 remark:@"修改权限" completion:^(int retcode, NSString \*errDescription) {

[weakSelf updateLogsText:[NSString stringWithFormat:@"修改二级用户权限%@ .retcode=%d,errText=%@",retcode==200?@"成功":@"失败",retcode,errDescription]];

}];

//管理员用户删除二级用户分享，按照指定设备targetKeyhash和二级用户authUserId（用户ID），可以删除设备的authUserId用户对targetKeyhash设备的控制权，即删除二级用户。

[self.zySdk unAuthorizeUserWithKeyhash:targetKeyhash authUserId:authUserId completion:^(int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"删除二级用户成功");

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"删除二级用户%@ .retcode=%d,errText=%@",retcode==200?@"成功":@"失败",retcode,errDescription]];

[weakSelf.zySdk getUserListWithKeyhash:targetKeyhash completion:^(NSMutableArray \*authUsers, NSMutableArray \*tempUsers, int retcode, NSString \*errDescription) {

NSString \*log=[NSString stringWithFormat:@"%d获取设备的用户列表%@,users=%@,tempUsers=%@,e=%@",retcode,retcode==200?@"成功":@"失败",authUsers,tempUsers,errDescription];

}];

}];

AuthorityTempEntity \*authority=[[AuthorityTempEntity alloc] init];

authority.timestamp1=now-6;//起点日期1

authority.timestamp2=now+86400;//结束日期2

authority.validType=0;//权限设置为时段访问

authority.shareTime=now;

//创建分享临时用户（临时用户无法在APP中使用，需要另外的网页——详情请z询问技术支持）

[self.zySdk shareTempUserWithKeyhash:targetKeyhash authority:authority remark:@"临时工1号" completion:^(NSString \*tempShareId, int retcode, NSString \*errDescription) {

[self updateLogsText:[NSString stringWithFormat:@"shareTemp%@=%@ retcode=%d,err=%@",retcode==200?@"成功":@"失败",tempShareId,retcode,errDescription]];

}];

NSString \*tempShareId=@"tempUserId";

authority.validType=1;//将临时用户的权限改为时段内单次

[self.zySdk modifyTempUserWithTempShareId:tempShareId authority:authority remark:@"test-临时" completion:^(int retcode, NSString \*errDescription) {

[self updateLogsText:[NSString stringWithFormat:@"modifyTemp%@=%@ retcode=%d,err=%@",retcode==200?@"成功":@"失败",tempShareId,retcode,errDescription]];

}];

## 获取用户下的设备列表

可以获取用户拥有的设备列表，代码示例：

//获取用户的设备列表，含一些特定属性，如在线状态、设备名称等

[self.zySdk getDevListWithSomeAttrsWithCompletion:^(NSMutableArray \*devs, int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"获取设备列表成功， devs=%@",devs);

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"getDevsWithAttrs .retcode=%d,devs=%@,errText=%@",retcode,devs,errDescription]];

}];

//获取用户的设备列表，不含任何属性值

[self.zySdk getDevListWithCompletion:^(NSMutableArray \*devs, int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"获取设备列表成功， devs=%@",devs);

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"getDevs .retcode=%d,devs=%@,errText=%@",retcode,devs,errDescription]];

}];

## 获取指定设备的所有属性

可以获取指定设备的所有属性值，代码示例：

//获取指定设备的所有属性值，targetKeyhash是指定设备的keyhash

[self.zySdk getDevAttrListWithKeyhash:targetKeyhash completion:^(DeviceInfoEntity \*dev, int retcode, NSString \*errDescription) {

if(retcode==200){

NSLog(@"获取设备属性all值成功 ，devInfo=%@",dev);

}

[weakSelf updateLogsText:[NSString stringWithFormat:@"getDevAttrs .retcode=%d,devs=%@,errText=%@",retcode,dev,errDescription]];

}];

## 控制设备

设置并实现指定Delegate代理，可以得到不同的回调结果。如代理MyEventResponseDelegate：发送Event后根据实现的onEventSyncResponseWithStatus回调接口查看发送结果，另外也可以通过实现onEventForAPP接口实时得到设备的状态变化，如设备上下线时的Event通知。

代码示例：

设置IOT相关代理代码示例：

//设置IOT Client状态变化监听代理：可以得到IOT Client的状态。当状态值为1或3时可以正常发送Event与设备通信。

[self.zySdk setIOTClientStateDelegate:self];

//设置event发送与接收的结果监听代理。与设备进行通信时的消息封装是Event。实现这个代理可以实时监听设备状态变化。如APP可以收到在Event[online@zot=0](表示设备掉线)时刷新UI.

[self.zySdk addZYEventResponseDelegate:self];

//设置IOT Client在线与异常离线的监听代理，当网络异常时client会断线重连，断线时发送Event会失败

[self.zySdk addZYClientOnlineDelegate:self];

代理实现监听IOT Client状态变化代码示例：

#pragma mark IOTOnlineDelegate

- (void)onIOTConnect{//IOT 连接成功时回调

NSLog(@"IOT Connect");

}

- (void)onIOTDisConnect{//IOT 异常断开时回调

NSLog(@"IOT DisConnect");

}

#pragma mark ZYIOTClientStateDelegate

-(void)onStopped{//IOT被手动stop时回调

NSLog(@"IOT stop.");

}

-(void)onStarted{//IOT被手动start时回调，表示正在start

NSLog(@"IOT start.");

}

-(void)onResume{//IOT被手动resume时回调

NSLog(@"IOT resume.");

}

-(void)onPaused{//IOT被手动pause时回调

NSLog(@"IOT pause.");

}

代理实现监听Event的发送和接收情况代码示例：

#pragma mark MyEventResponseDelegate

/\*\*

接收到一个event消息，此时Event接收方是APP。ZY平台主动下发event时source无值。

@param event 消息封装，包含设备属性名、属性值等

@param source 发出Event的一方，指的是设备keyhash或null(平台)。如设备状态变更时发送Event通知APP，此时source是设备keyhash.

@param isRead 表示本Event是不是Event读属性(sendEventToReadDevAttrFromAPP)的回复

@param readState 表示Event读属性结果是否成功

\*/

-(void)onEventForAPP:(MyEventData \*)event fromSource:(NSString \*)source isRead:(BOOL)isRead readState:(BOOL)readState {}

/\*\*接收到一个event消息，此时接收方是app的直连子设备(不直接使用IOT的设备是子设备，对于APP来说子设备是APP直连子设备比如蓝牙设备，另外还有网关下挂的子设备)。监听到本event需要将值再发送给对应的子设备。

@param event Event消息封装，包含设备属性名、属性值等

@param source 发出这个Event的一方（表示对应的设备ID或null）

@param childDevKeyhash 子设备ID（表示需要将这个Event的属性值再发给对应的子设备），如现用户的设备列表有一个蓝牙子设备C，此时接收到本Event[attrName=statew,type=1,attrValue1=2]且childDevKeyhash=C，那么APP通过蓝牙为C设备设置statew值为2。

@param isRead 表示本Event是不是Event读属性(sendEventToReadDevAttrFromChildDev)的回复

@param readState 表示Event读属性结果是否成功

\*/

- (void)onEventForChildDev:(MyEventData \*)event fromSource:(NSString \*)source toChildDevKeyhash:(NSString \*)childDevKeyhash isRead:(BOOL)isRead readState:(BOOL)readState {}

/\*\*

event发送结果的回调（发送到服务器不代表控制设备成功，应以设备实际属性值为准）

@param isSuccess true表示成功发送到IOT服务器，false表示失败

@param eventId 发送Event时指定的或返回的消息ID

\*/

-(void)onEventSyncResponseWithStatus:(BOOL)isSuccess eventId:(int32\_t)eventId{}

/\*\*子设备登录和退出登录时的结果回调

@param loginResult 登录或退出登录的结果

@param isLogin 表示登录或退出登录，true表示登录，false表示退出登录。

@param childDevKeyhash 是这个子设备的登录（退出登录）

@param childDevType 本子设备的类型（如蓝牙或串口，分别是bluetooth和com，都是小写字母）

\*/

-(void)onEventForChildDevLoginOrOut:(BOOL)loginResult isLogin:(BOOL)isLogin childDevKeyhash:(NSString \*)childDevKeyhash childDevType:(NSString \*)childDevType{}

//发送event后，接收event失败。eventId是消息ID，发送event时可以指定

-(void)onEventFailureWithEventId:(int32\_t)eventId{

[self updateLogsText:[NSString stringWithFormat:@"\tevent failure=%d",eventId]];

}

APP发送Event代码示例：

//APP发送Event到设备，属性值类型是String

ret=[self.zySdk sendEventStrFromApp:10 attrName:attrName valueStr:attrValue1 targetKeyhash:targetKeyhash];

//APP发送Event到设备，属性值类型是int

ret=[self.zySdk sendEventIntFromApp:2 attrName:attrName valueInt:attrV2 targetKeyhash:targetKeyhash];

//APP发送Event到设备，属性值类型是hex(如语音文件的NSData内容)

NSData \*data=data0;

ret=[self.zySdk sendEventHexFromApp:3 attrName:@"hextest" valueHexBytes:data targetKeyhash:targetKeyhash];

## 设备固件升级

调用ZYIOTSDK中ZYFotaAPI的接口检测设备的固件版本更新情况，并实现对应的监听接口获取从筑云返回的版本状态，然后决定是否要升级。

代码示例：

检测新固件版本：

//检测设备版本是否需要升级

ZYFotaAPI \*devFotaApi=[ZYFotaAPI getFotaInstance];//初始化接口实例

if([devFotaApi connectStatus]!=1){//调用接口前定要先判断连接Fota的状态

[devFotaApi openConnection];

}

NSLog(@"fota connectStatus=%d",[devFotaApi connectStatus]);

//指定设备targetKeyhash进行检测是否升级，并指定结果回调接口

[devFotaApi setDevUpdateWithDelegate:self keyhash:targetKeyhash];

[devFotaApi checkDevUpdateVersion];//检测设备版本

得到版本信息决定升级：

#pragma mark -->ZYIOT manager delegate end.

#pragma Fota delegate (retcode==1 is ok)

/\*\*获取设备固件升级进度

@param retcode retcode=1时成功，其余数字表示失败

@param keyhash keyhash fota模块当前keyhash

@param isNewest isNewest true表示本设备已经是最新的设备

@param isForceUpdate 本设备固件的当前版本若不是最新版本，固件不升级是否可用。true表示固件不升级就无法正常使用。

@param currentV 本设备的当前版本

@param newestV 本设备对应的最新的设备固件

\*/

- (void)callCheckDevUpdateWithRetcode:(int16\_t)retcode keyhash:(NSString \*)keyhash isNewest:(BOOL)isNewest isForceUpdate:(BOOL)isForceUpdate currentVer:(NSString \*)currentV newestVer:(NSString \*)newestV {

NSLog(@"升级时回调 retcode=%d, isNewest=%d isForce=%d, keyhash=%@",retcode,isNewest,isForceUpdate,keyhash);

if(retcode==1){//成功获取版本信息

if(isNewest==NO)

{

NSString \*result=[[ZYFotaAPI getFotaInstance] setDevUpdateStart];//通知设备升级（是通过Event通知设备升级的，由SDK内部发送消息）

NSLog(@"result=%@",result);

if([result intValue]!=0){//发出升级命令

[[ZYFotaAPI getFotaInstance] getDevUpdateProgress];//获取设备升级进度（通知设备升级成功后，才有进度值）

}

}

}

}

/\*\*获取设备固件升级进度

@param retcode retcode=1时成功，其余数字表示失败

@param keyhash keyhash fota模块当前keyhash

@param progress 设备固件升级的进度

\*/

- (void)callGetDevUpdateProgressWithRetcode:(int16\_t)retcode keyhash:(NSString \*)keyhash progress:(int)progress {

NSLog(@"percentage升级时回调进度 retcode=%d, keyhash=%@ per=%d",retcode,keyhash,progress);

if(retcode==1){//成功获取版本升级进度

//continue get progress ? 如果需要持续接收进度，可以继续调用获取进度的接口

if(progress!=100){

NSLog(@"继续获取进度");

dispatch\_after(dispatch\_time(DISPATCH\_TIME\_NOW, (int64\_t)(2 \* NSEC\_PER\_SEC)), dispatch\_get\_main\_queue(), ^{

[[ZYFotaAPI getFotaInstance] getDevUpdateProgress];

});

}

}

}

//连接Fota模块断开

- (void)onCloseConnectToFota {

NSLog(@"检测升级结束-----");

}

#pragma Fota delegate end

## 关闭IOT Client

当SDK不需要与设备进行通信时，选择暂停或关闭IOT Client，可以降低通信损耗。

初始化SDK和获取设备列表时，会自动启用IOT Client。

//需要与设备进行通信时，若getIOTClientState值不是1或3，可以选择以下接口手动启动IOT Client。

[self.zySdk toASyncStartOrStopIOTClient:1];

//不需要与设备进行通信时可以暂时断开IOT Client

[self.zySdk toASyncStartOrStopIOTClient:2];

//彻底不需要与设备进行通信时可以彻底断开IOT Client，如退出APP时

[self.zySdk toASyncStartOrStopIOTClient:-1];

## 获取服务器HttpDNS信息

获取HttpDNS信息，APP版本不同可选择使用不同的DNS服务器。

[self.zySdk setHttpDNSServerDomain:@"zhuyun.f3322.net"];

NSLog(@"your dnskey=%@",[self.zySdk getDNSKey]);

// //DNSKey是由筑云分配给APP开发者的，请提前申请

[self.zySdk setDNSKey:@"your DNSKey"];

// [self.zySdk setDNSKey:@"5d0728741313f1bc384b9dbb"];

//调用此接口前请先调用SDK接口setDNSKey,设置DNSKey之后才可以成功获取APP对应的服务器DNS信息

[self.zySdk getHttpDNSInfosWithCompletion:^(NSMutableArray \*httpDNSInfos, int retcode, NSString \*errDescription) {

//此处获取到了APP不同版本的服务器DNS信息

NSString \*log=[NSString stringWithFormat:@"%d getHttpDNSInfo结果=%@,e=%@",retcode,httpDNSInfos,errDescription];

[self updateLogsText:log];

}];