**ZYIOT Account SDK**

**使用说明**

版本：V2.2 日期：2019-06-13

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** |
| 2019-06-13 | V2.2.0 | 1. 增加Account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 |
| 2019-06-06 | V2.1.0 | 增加Account用户相关接口 |
| 2018-12-27 | V2.0.0 | 1. 新增网关子设备功能 2. 新增Open Account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](#_Toc10795)

[2 简介 4](#_Toc8361)

[3 名词解释 5](#_Toc1852)

[4 功能介绍 7](#_Toc205)

[5 配置SDK 8](#_Toc10728)

[5.1 环境准备 8](#_Toc21537)

[5.2 SDK安装 8](#_Toc28684)

[5.3 配置工程 8](#_Toc27387)

[6 主要流程 11](#_Toc29384)

[6.1 初始化SDK 11](#_Toc894)

[6.2 手机账号登录 11](#_Toc25398)

[6.3 绑定设备、分享设备、编辑被分享权限、删除设备 12](#_Toc12275)

[6.4 获取用户下的设备列表 14](#_Toc25026)

[6.5 获取指定设备的所有属性 15](#_Toc12592)

[6.6 控制设备 15](#_Toc19841)

[6.7 设备固件升级 21](#_Toc27951)

[6.8 关闭IOT Client 22](#_Toc13042)

# 接入必读

Android的demo已经包含了开放平台SDK的jar文件，请将SDK的jar文件拷贝到项目工程的libs文件夹中。

# 简介

本文档用于说明筑云开放平台SDK Android版本接口之间的关系以及接口调用顺序，对开放平台SDK Android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\_android.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 |
| 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） |
| 设备套餐 | 可以获取设备套餐信息，下单购买套餐 |
| 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

# 配置SDK

* 1. 环境准备

支持 Android Studio 1.5 以上 支持 JDK 7.0 以上版本  
 支持 Android 手机系统 4.2 以上版本

* 1. SDK安装

1. 解压缩ZYIOTSDK.zip；
2. 拷/libs下jar包到自己工程对应目录（包含下图中5个jar包）；

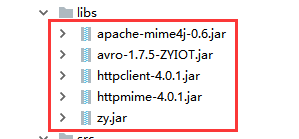


图5-1

1. 将以下依赖内容添加到build.gradle文件 的dependencies内：

implementation **'com.android.support:multidex:1.0.3'**implementation fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])  
implementation **'org.lucee:commons-io:2.6.0'**implementation **'commons-codec:commons-codec:1.9'**implementation **'org.apache.commons:commons-compress:1.18'**implementation **'org.apache.httpcomponents:httpcore:4.4.10'**implementation **'org.glassfish:javax.annotation:10.0-b28'**implementation **'org.slf4j:log4j-over-slf4j:1.7.25'**implementation **'ch.qos.logback:logback-classic:1.1.2'**implementation **'ch.qos.logback:logback-core:1.1.2'**implementation **'org.slf4j:slf4j-api:1.7.25'**implementation **'org.codehaus.jackson:jackson-core-asl:1.9.13'**implementation **'org.codehaus.jackson:jackson-mapper-asl:1.9.12'**implementation **'org.bouncycastle:bcprov-jdk15on:1.60'**implementation **'io.netty:netty-all:4.1.6.Final'**

* 1. 配置工程

1. 配置 Android 权限

在 AndroidMainfest.xml 文件中添加：

<**uses-permission android:name="android.permission.INTERNET"** />  
<**uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"** />  
<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />  
<**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />

1. 配置Android 使用Http

在 AndroidMainfest.xml 文件中添加：允许使用http请求、允许使用Apache Http。

<**application  
 ……** android:usesCleartextTraffic="true"  
 >  
 <**uses-library android:name ="org.apache.http.legacy" android:required ="false"**/>  
</**application**>

1. 配置build.gradle

defaultConfig {

……  
 multiDexEnabled **true** *//to deal with too much jar code*}

buildTypes {  
 release {

**……** lintOptions {*//mutil dex* checkReleaseBuilds **true** abortOnError **false** }  
 }  
 debug {

……lintOptions {*//mutil dex* checkReleaseBuilds **false** abortOnError **false** }  
 }  
}

packagingOptions {  
 exclude **'META-INF/DEPENDENCIES'** exclude **'META-INF/ASL2.0'**}

注意：因代码量过多，所以需要引入multiDex功能包，并为其配置build.gradle。

1. 配置 AndroidManifest.xml

添加如下配置，指定启动的Application，用于引入multiDex功能。其中yourSelfApplication替换为自己代码中定义的Application类——自定义Application一定要继承MultiDexApplication。

<**application  
 android:name="yourSelfApplication"  
 ……**

**>**

**</application>**

## 

# 主要流程

## 初始化SDK

*//获取ZYSDK实例***zySdk** = ZYAccountSDK.*getZYAccountSDKInstance*(getApplicationContext());  
*//指定SDK使用的tenantId***zySdk**.setTenantId(1);  
*// 默认的服务器有效，不需要修改也可以。IOT、ZOT、FOTA服务器域名指向有效，默认的服务器指向是有效的，可以不用改，（如果重新部署迁移服务器，需要重新set）***zySdk**.setZotDomain(**"zhuyun.f3322.net"**);*//ZOT模块服务器IP入口***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.*getInstance*().setFotaDomain(**"zhuyun.f3322.net"**);*//设置FOTA服务器是单IP入口*  
String phoneKeyhash = **zySdk**.getPhoneKeyhash();*//获取本机的keyhash*System.***out***.println(**"我自己的keyhash="**+phoneKeyhash);

## 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。

**final** String phone=**etUserAccount**.getText().toString();  
**final** String pwd=**etUserPwd**.getText().toString();  
*//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***zySdk**.login(phone, pwd, **null**, **null**, **null**, **new** ZYListener.getUserToken() {  
 @Override  
 **public void** callBackUserToken(ZYUserToken zyUserToken, **int** retcode, String errDescription) {  
 *//登录结果* String result=phone+**"登录结果retcode="**+retcode+**",token="**+zyUserToken+**",err="**+errDescription;  
 appendLogs(result);  
 }  
 });

## 绑定设备、分享设备、编辑被分享权限、删除设备

支持绑定设备、删除设备、分享设备、编辑设备用户权限等。

代码示例：

绑定设备：

*/\*\*  
 \* 用户绑定设备（扫描设备二维码得到keyhash ,devTypeId），  
 \* 设备只允许一个用户绑定，绑定后该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.  
 \*/***private void** bindDev(String keyhash, String devTypeId) {  
 **zySdk**.bindDev(keyhash, devTypeId, **new** ZYListener() {  
 @Override  
 **public void** callBackRetcode(**int** retcode, String errDescription) {  
 appendLogs(retcode + **" 绑定设备结果.isOK="** + (retcode == 200) + **" err="** + 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。  
 \*/***private void** deleteDev(String keyhash) {  
 **zySdk**.deleteDevice(keyhash, **new** ZYListener() {  
 @Override  
 **public void** callBackRetcode(**int** retcode, String errDescription) {  
 appendLogs(retcode + **" 删除设备结果.isOK="** + (retcode == 200) + **" err="** + errDescription);  
 }  
 });  
}

分享设备、删除被分享的用户，以及编辑分享的用户权限：

*/\*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（已在筑云注册的手机号或邮箱）；  
authority为空时表示给予全时访问即每一天每时刻都允许控制设备（具体其他权限详见API ZYAccountSDK接口文档），remark是备注内容可以是昵称。\*/***private void** shareDevAboutApi(String keyhash) {  
 String account =**"12345678999"**;*//分享给指定用户* **long** nowSecond = System.*currentTimeMillis*() / 1000;  
 *//权限：时段访问，时间限制：起始-结束时间戳(秒)。*

*//封装示例：* AuthorityEntity1 authority1 = **new** AuthorityEntity1();*//用户权限封装：时段访问* authority1.setValidType(1);*//权限：时段访问* authority1.setTimestamp1(nowSecond / 1000);*//起始时间戳(秒)* authority1.setTimestamp2(nowSecond / 1000 + 300);*//结束时间戳(秒)*

*//将设备分享给指定的用户，account是已在筑云注册的手机或邮箱。此时指定被分享的用户是这个设备的二级用户。  
// 如果传入的权限为空的话，则表示给予二级用户全时访问权限。* **zySdk**.authorizeUser (keyhash, account, authority1, **"i'm auth-remark"**, **new** ZYListener.authToUser() {  
 @Override  
 **public void** callBackAuthUser(ZYAuthUser authUser,**int** retcode, String errDescription) {  
 appendLogs(retcode + **" auth结果.isOK="** + (retcode == 200) +**",user="**+authUser+ **" err="** + errDescription);  
 }  
 });

*//设备管理员可删除二级用户(被分享用户)对于设备的权限* **zySdk**.unAuthorizeUser(keyhash, **"二级用户的UserId"**, **new** ZYListener() {  
 @Override  
 **public void** callBackRetcode(**int** retcode, String errDescription) {  
 appendLogs(retcode + **" auth delete 设备结果.isOK="** + (retcode == 200) + **" err="** + errDescription);  
 }  
 });  
 AuthorityEntity2 authority2=**new** AuthorityEntity2();*//定期访问* authority2.setTime1(**"10:00"**);*//上午10点整* authority2.setTime2(**"14:45"**);*//下午2点45分* authority2.setWeekMode(**"12"**);*//允许周一和周二的时段  
 //设备管理员可以修改二级用户的权限* **zySdk**.modifyAuthUserForEntity(keyhash, **"二级用户的UserId"**, authority2 , **"修改二级用户权限"**, **new** ZYListener() {  
 @Override  
 **public void** callBackRetcode(**int** retcode, String errDescription) {  
 appendLogs(retcode + **" 修auth设备结果.isOK="** + (retcode == 200) + **" err="** + errDescription);  
 }  
 });  
  
 *//获取设备下的用户,得到与指定设备有关联的所有用户，包括APP用户(一级用户和二级用户)、临时用户。* **zySdk**.getUserList(keyhash, **new** ZYListener.getUserList() {  
 @Override  
 **public void** callBackUserList(List<ZYAuthUser> authUsers, List<ZYTempUser> tempUsers, **int** retcode, String errDescription) {  
 appendLogs(retcode + **" 获取设备下的用户列表.err="** + errDescription + **" UL="** + authUsers + **" tempUL="** + tempUsers);  
 }  
 });  
}

## 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：

*/\*获取账号下的设备列表 \*/***private void** getDevList() {  
 *//用户获取其下的设备列表，不含任何属性值* **zySdk**.getDevList(**new** ZYListener.getDevList() {  
 @Override  
 **public void** callBackDevList(List<DeviceInfoEntity> devs, **int** retcode, String errDescription) {  
 appendLogs(retcode + **" 获取设备列表结果="** + devs);  
 }  
 });  
 *//用户获取其下的设备列表，含一些特定属性，如在线状态、设备名称等* **zySdk**.getDevListWithSomeAttrs(**new** ZYListener.getDevList() {  
 @Override  
 **public void** callBackDevList(List<DeviceInfoEntity> devs, **int** retcode, String errDescription) {  
 System.***out***.println(retcode + **" 获取设备列表(带些属性)结果="** + devs);  
 }  
 });  
}

## 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\*/***private void** getDevAttrs(String keyhash) {  
 *// 用户获取指定设备的所有属性值* **zySdk**.getDevAttrList(keyhash, **new** ZYListener.getDevInfo() {  
 @Override  
 **public void** callBackDevInfo(DeviceInfoEntity dev, **int** retcode, String errDescription) {  
 appendLogs(retcode + **" 获取设备属性结果="** + dev);  
 }  
 });  
 *// 用户获取指定设备不包含全局属性的所有属性值* **zySdk**.getDevAttrListForNoGlobal(keyhash, **new** ZYListener.getDevInfo() {  
 @Override  
 **public void** callBackDevInfo(DeviceInfoEntity dev, **int** retcode, String errDescription) {  
 System.***out***.println(retcode + **" 获取设备no global属性结果="** + dev);  
 }  
 });  
}

## 控制设备

设置并实现指定Listener接口，可以得到不同的回调结果。如实现ZYEventResponseListener：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。代码示例如下：

设置回调接口（代理）并实现：

*/\*此时监听代理回调时都指向本类\*/  
//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。***zySdk**.setIOTClientStateListener(**this**);  
  
*//设置event发送与接收的结果监听代理。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.***zySdk**.addZYEventResponseListener(**this**);  
  
*//设置IOT Client在线与异常离线状态的监听代理，当网络异常时client会断线重连，断线时发送Event会失败***zySdk**.addZYClientOnlineListener(**this**);  
  
*//设置监听SDK的Login状态，可及时知道SDK登录过期的状态，然后决定是否手动initSDK***zySdk**.setSDKLoginStateListener(**this**);

实现接口IOTOnlineListener：代理实现监听IOT Client状态变化

*/\* Online.IOTOnlineListener start\*/*@Override  
**public void** onIOTConnect() {*//IOT 连接成功时回调（此后可以正常收发Event）* String log = **"ZYIOT Connect over"**;  
 appendLogs(log);  
}  
  
@Override  
**public void** onIOTDisConnect() {*//IOT 连接出错或断开时回调（此后无法收发Event）* String log = **"ZYIOT Disconnect exp"**;  
 appendLogs(log);  
}  
*/\* Online.IOTOnlineListener end\*/*

实现接口IOTClientStateListener：代理实现监听IOT Client状态变化

*/\* ZYIOTClientStateListener start\*/*@Override  
**public void** onStarted() {*//IOT被手动start时回调，表示正在start* appendLogs(**"IOT on Start"**);  
}  
  
@Override  
**public void** onStopped() {*//IOT被手动stop时回调，表示正在stop* appendLogs(**"IOT on Stopped"**);  
}  
  
@Override  
**public void** onPaused() {*//IOT被手动pause时回调，表示正在pause* appendLogs(**"IOT on Paused"**);  
}  
  
@Override  
**public void** onResumed() {*//IOT被手动resume时回调，表示正在resume* appendLogs(**"IOT on Resume"**);  
}  
  
@Override  
**public void** onStartFailure(ZYIOTException exception) {*//IOT被手动start时出错回调* appendLogs(**"IOT on Start failure"**);  
}  
  
@Override  
**public void** onPauseFailure(ZYIOTException exception) {*//IOT被手动pause时出错回调* appendLogs(**"IOT on Paused failure"**);  
}  
  
@Override  
**public void** onResumeFailure(ZYIOTException exception) {*//IOT被手动resume时出错回调* appendLogs(**"IOT on Resume Failure"**);  
}  
  
@Override  
**public void** onStopFailure(ZYIOTException exception) {*//IOT被手动stop时出错回调* appendLogs(**"IOT on Stop failure"**);  
}  
*/\* ZYIOTClientStateListener start\*/*

实现接口ZYEventResponseListener：代理实现监听Event的发送和接收情况

*/\*\* ZYEventResponseListener start\*/  
  
 /\*\*  
 \* 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 eventData*** *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*** *是读属性的Event回复，true是响应读，否则是普通Event  
 \** ***@param readState*** *Event读属性结果，true成功，否则失败  
 \*/* @Override  
 **public void** onEventForAPP(ZyEventData eventData, String source, **boolean** isRead, **boolean** readState) {  
 String log = **null**;  
 **int** eventId = eventData.getEventId();*//消息ID，可在发送Event时指定eventId* String attrName = eventData.getAttrName();*//属性名* **int** attrVaType = eventData.getAttrType();*//属性值的类型，根据不同的属性值类型解析不同的value字段，得到属性值。如type=1时，attrValue1为属性值。* String attrV = attrVaType == 1 ? **"strV="** + eventData.getAttrValue1() : **"intV="** + eventData.getAttrValue2();  
 **if** (isRead && readState == **false**) {  
 log = (**hadReceivedResponseNumber**++) + **" app读取属性失败eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" source="** + source;  
 appendLogs(log);  
 **return**;  
 } **else if** (isRead) {  
 log = (**hadReceivedResponseNumber**++) + **" app读取属性成功eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" attrV="** + attrV + **" source="** + source;  
 } **else** {  
 log = (**hadReceivedResponseNumber**++) + **"app收到eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" attrV="** + attrV + **" source="** + source;  
 }  
 appendLogs(log);  
 parseHexFrames(eventData, source);  
 }  
  
 */\*\*  
 \* 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 eventData*** *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*** *是读属性的Event回复，true是响应读，否则是普通Event  
 \** ***@param readState*** *Event读属性结果，true成功，否则失败  
 \*/* @Override  
 **public void** onEventForChildDev(ZyEventData eventData, String source, String childDevKeyhash, **boolean** isRead, **boolean** readState) {  
 String log = **null**;  
 **int** eventId = eventData.getEventId();*//消息ID，可在发送Event时指定eventId* String attrName = eventData.getAttrName();*//属性名* **int** attrVaType = eventData.getAttrType();*//属性值的类型，根据不同的属性值类型解析不同的value字段，得到属性值。如type=1时，attrValue1为属性值。* String attrV = attrVaType == 1 ? **"strV="** + eventData.getAttrValue1() : **"intV="** + eventData.getAttrValue2();  
 **if** (isRead && readState == **false**) {  
 log = (**hadReceivedResponseNumber**++) + **" child读取属性失败eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" source="** + source;  
 appendLogs(log);  
 **return**;  
 } **else if** (isRead) {  
 log = (**hadReceivedResponseNumber**++) + **" child读取属性成功eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" attrV="** + attrV + **" source="** + source;  
 } **else** {  
 log = (**hadReceivedResponseNumber**++) + **"child收到eventId="** + eventId + **" event.attrName="** + attrName + **" vaType="** + attrVaType + **" attrV="** + attrV + **" source="** + source;  
 }  
 appendLogs(log);  
 parseHexFrames(eventData, source);  
 }  
  
 */\*\*  
 \* 子设备登录和退出登录时的结果回调  
 \*  
 \** ***@param isSuccess*** *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 devType*** *子设备的类型（如蓝牙或串口，分别是bluetooth和com，都是小写字母）  
 \** ***@param childKeyhash*** *是这个子设备的登录（退出登录）  
 \*/* @Override  
 **public void** onEventForChildDevLoginOrOut(**boolean** isSuccess, **boolean** isLogin, String devType, String childKeyhash) {  
 appendLogs(**"收到childDev Login请求结果：onEventChildLogin isOK="** + isSuccess + **" ;isLogin="** + isLogin + **" type="** + devType + **" childKeyhash source="** + childKeyhash);  
  
 }  
  
 */\*\*  
 \* event发送结果的回调（当Event发送出去后，到达IOT模块时会有这个回调）。  
 \* 小概率情况下即使到达IOT也无法到达设备，因为ZOT有鉴权token。所以请在发Event时保证SDKLogin的状态正常。  
 \*  
 \** ***@param isSuccess*** *true表示成功发出Event到达IOT，false表示失败  
 \** ***@param eventId*** *发送Event时指定的或返回的消息ID  
 \*/* @Override  
 **public void** onEventSyncResponseWithStatus(**boolean** isSuccess, **int** eventId) {  
 appendLogs((**hadReceivedResponseNumber**++) + **" 发出：status="** + isSuccess + **" eventId="** + eventId);  
 }  
  
 @Override  
 **public void** onEventFailureWithEventId(**int** eventId) {  
 *//（极少少出现）发送event后，接收event失败。eventId是消息ID，发送event时可以指定* appendLogs(**"收到 event 失败： eventId="** + eventId);  
 }  
*/\*\*ZYEventResponseListener end\*/*

实现接口ZYSDKLoginStateListener：代理实现监听SDK的Login状态是否有效

*/\*\*ZYSDKLoginStateListener start\*/* @Override  
 **public void** sdkLoginSuccess() {  
 appendLogs(**"SDK Login Success"**);  
 }  
  
 @Override  
 **public void** sdkLoginFailure(**int** retcode, String errDescription) {  
 appendLogs(**"SDK login state exp{"** + retcode + **"}:正在"** + errDescription);  
 login();*//SDK登录状态不正确，如果不重新登录则无法使用API(因为筑云Token无效)* }  
  
 */\*\*ZYSDKLoginStateListener end\*/*

APP发送Event：

*//APP发送Event到设备，属性值类型是String* logText = **zySdk**.sendEventStrFromApp(11, attrName, valueStr, keyhashTarget);  
  
*//APP发送Event到设备，属性值类型是int* logText = **zySdk**.sendEventIntFromApp(12, attrName, valueInt, keyhashTarget);  
  
*//APP发送Event到设备，属性值类型是hex(如语音文件的byte[]内容)* vaHex=valueStr.getBytes();  
 logText = **zySdk**.sendEventHexFromApp(14, attrName, vaHex, keyhashTarget);

## 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：

*//指定设备keyhash进行检测是否升级，并指定结果监听回调接口(fota一次只允许检测一个设备)*ZYFotaAPI.*getInstance*().setDevUpdateInfo(keyhashTarget, **this**);  
*//检测设备版本是否需要升级*ZYFotaAPI.*getInstance*().checkDevUpdateVersion();

得到版本信息决定升级：

*/\*ZYDevUpdateListener start\*/  
  
/\*\*  
 \* 获取设备固件版本信息结果回调  
 \*  
 \** ***@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  
 \*/***public void** callCheckDevUpdate(**int** retcode, String keyhash, **boolean** isNewest, **boolean** isForceUpdate, String currentV, String newestV) {  
  
 appendLogs(retcode + **" 检测设备版本="** + isNewest + **" v="** + newestV + keyhash);  
 *//retcode为1时表示获取设备版本信息成功* **if** (retcode == 1 && !isNewest) {  
 *//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），当返回的是eventId时才表示正常发起命令(实际结果详看onEventSyncResponseWithStatus收到的对应EventId的status)，其余的String描述均表示未发出命令* String eventId = ZYFotaAPI.*getInstance*().setDevUpdateStart();  
 **if** (parseStrGetInt(eventId) > 0) {*//发送命令成功* appendLogs(**"发送命令设备升级固件="** + eventId);  
 ZYFotaAPI.*getInstance*().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结果回调  
 \*  
 \** ***@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  
 \*/***public void** callGetDevUpdateProgress(**int** retcode, String keyhash, **int** progress) {  
 appendLogs(retcode + **" 设备进度="** + progress + **" v="** + keyhash);  
 **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* }  
}  
  
@Override  
**public void** onCloseConnectToFota() {*//当fota与服务器的连接断开时回调* appendLogs(**" Fota 连接 断开或结束"**);  
}  
*/\*ZYDevUpdateListener 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。

*//IOTClient建立连接***zySdk**.toASyncStartOrStopIOTClient(1);  
  
*//不需要与设备进行通信时可以暂时断开IOT Client。（程序切到后台建议调用toASyncStartOrStopIOTClient(2)）***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***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。

*//DNSKey是由筑云分配给APP开发者的，请提前申请*appendLogs(**"当前SDK内的DNSKey="**+**zySdk**.getDNSKey());  
**if**(**zySdk**.getDNSKey()==**null**){  
 **zySdk**.setDNSKey(**"5d0728741313f1bc384b9dbb"**);  
}  
**zySdk**.setHttpDNSServerDomain(**"zhuyun.f3322.net"**);  
*//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***zySdk**.getHttpDNSInfo(**new** ZYListener.getHttpDNSInfoList() {  
 @Override  
 **public void** callBackDNSInfoList(List<HttpDNSInfo> list, **int** retcode, String errDescription) {  
 *//此处获取到了APP不同版本的服务器DNS信息* appendLogs(retcode+**"获取DNS信息="**+list+**" ,err="**+errDescription);  
 }  
});