GizWifiSDK API Android 参考手册

修订记录

修改时间	修改内容	版本	修改人	备注
2016.5.10	更新目录	0.9.0	Pomia	
2016.6.14	校对	1.0.0	Pomia	
2016.9.7	增加定时任务接口	1.0.1	Pomia	
2016.9.27	增加新接口说明	1.0.2	Pomia	
2016.10.19	启动接口中增加域名、pk 过滤参数 设备配置时的模组类型增加一个自定义枚举值 旧的启动接口仍然兼容，但不推荐使用 旧的切换域名接口仍然兼容，但不推荐使用 定时任务接口已废弃，不推荐使用	1.0.3	Pomia	
2016.11.7	启动接口参数使用变更 增加设备全球域名部署接口	1.0.4	Pomia	
2016.11.30	startWithAppID 接口增加开启设备域名自动设置参数 setDeviceServerInfo 接口 mac 参数使用变更	1.0.5	Pomia	
2017.1.25	增加新的设备定时任务接口 增加设备分享接口 增加一些枚举定义	1.0.6	Pomia	
2017.4.13	增加新的添加子设备接口	1.0.7	Pomia	
2017.8.14	增加用户反馈接口	1.0.8	Pomia	
2017.8.15	修改启动接口功能描述	1.0.9	Pomia	
2017.9.30	增加新的启动接口	1.1.0	Pomia	

	新增设备 OTA 接口、中控分组、场景接口 补充错误码、枚举定义			

1. GizWifiSDK 类

1.1. 简介

机智云 Wifi SDK 的基础类，为 APP 开发者提供设备配置和发现、设备分组、用户登录和注册等功能。

1.2. 属性方法

属性方法名	定义
setListener	public void setListener(GizWifiSDKListener listener)
getDeviceList	public List<GizWifiDevice> getDeviceList()

1.3. 回调接口

以下是 GizWifiSDK 提供的所有回调接口，将在在后续 API 定义中详细介绍：

- didNotifyEvent：SDK 系统事件通知
- didGetCurrentCloudService：服务域名独立部署的回调接口
- didDisableLAN：小循环是否禁用的回调接口
- didDiscovered：设备列表上报的回调接口
- didGetSSIDList：获取设备周围 Wi-Fi 热点列表的回调接口
- didSetDeviceOnboarding：设备配置结果的回调接口
- didBindDevice：设备绑定结果的回调接口
- didUnbindDevice：设备解除绑定结果的回调接口
- didUpdateProduct：设备配置文件上报的回调接口
- didCreateScheduler：创建定时任务的回调接口
- didDeleteScheduler：删除定时任务的回调接口
- didGetSchedulers：获取定时任务列表的回调接口

- didGetSchedulerStatus：查询定时任务执行状态的回调接口
- didGetCaptchaCode：获取图片验证码的回调接口
- didRequestSendPhoneSMSCode：请求手机短信验证码的回调接口
- didVerifyPhoneSMSCode：验证手机短信验证码的回调接口
- didRegisterUser：用户注册结果的回调接口
- didUserLogin：用户登录结果的回调接口
- didTransAnonymousUser：匿名用户转换的回调接口
- didChangeUserPassword：更换用户密码结果的回调接口
- didChangeUserInfo：修改用户信息结果的回调接口
- didGetUserInfo：获取用户信息的回调接口
- didGetGroups：获取用户设备分组列表的回调接口

1.4. API 定义

【sharedInstance】

定义	public static synchronized GizWifiSDK sharedInstance()
功能描述	获取 GizWifiSDK 单例的实例。
返回值	SDK 唯一的实例。
代码示例	GizWifiSDK mSDKInstance = GizWifiSDK.sharedInstance();

【setListener】

定义	public void setListener(GizWifiSDKListener listener)
功能描述	设置 SDK 通用监听器

参数	listener	GizWifiSDKListener 回调对象

代码示例	GizWifiSDK.sharedInstance().setListener(new GizWifiSDKListener() {
	// app 根据自己的需要实现回调函数
	});	

【startWithAppInfo】

定义	public void startWithAppInfo(Context context, <ConcurrentHashMap<String, String> appInfo, List<ConcurrentHashMap<String, String>> productInfo, ConcurrentHashMap<String, String> cloudServiceInfo, boolean autoSetDeviceDomain)
功能描述	初始化 SDK。该接口执行后，其他接口功能才能正常执行。如果已经设置了 delegate，SDK 会立即通过 didDiscovered 上报发现的设备。 如果 App 要做域名切换和设备的 productKey 过滤，建议在 SDK 初始化时就指定好要切换的域名和产品 productKey。 如果需要设置设备连接的云服务域名，可以在该接口调用时开启自动设置功能。SDK 会为所有已与 AppID 关联的设备设置域名，支持域名设置的设备会与 App 连接到同一个云服务域名上。但该接口默认是不开启此功能的。 注意：设备域名自动设置开启后会一直生效，但调用 setDeviceServerInfo 接口时将会终止自动设置

参数	context	上下文对象
	appInfo	应用信息，格式：{"appId": "xxx", "appSecret": "xxx"}。此参数不能填 nil，appId 和 appSecret 必须为有效值。 在机智云开发者中心 dev.gizwits.com 中，每个注册的设备在对应的“应用配置”中，都能够查到对应的 appId 和 appSecret
	productInfo	产品信息数组，格式：[{"productKey": "xxx", "productSecret": "xxx"}]，此参数为选填。 如果填写了此参数，需保证 productKey 和 productSecret 都为有效值，否则会被忽略。SDK 会根据此参数过滤设备列表

	cloudServiceInfo	服务器域名信息。
		如果使用机智云统一部署的云服务域名，此参数填 nil，此时将根据用户手机的地理位置信息使用匹配的域名。
		独立部署时此参数必须指定域名信息，形如：xxx.gizwits.com
		如果要使用特殊端口号，需同时指定 Http 和 Https 端口：xxx.gizwits.com:81&8443。
		参数为字典格式：
		{
		"openAPIInfo": "xxx", // NSString类型，api服务域名，必填
		"siteInfo": "xxx" // NSString类型，site服务域名，可不填
		"pushInfo": "xxx" // NSString类型，推送服务域名，不用不用填
		}

	autoSetDeviceDomain	是否要开启设备域名的自动设置功能。此参数默认值为 false，即不开启自动设置。 参数值传 true，则开启设备域名的自动设置功能。如果开启了设备域名的自动设置，小循环设备将被连接到 App 当前使用的云服务域名上
回调	public void didNotifyEvent(GizEventType eventType, Object eventSource, GizWifiErrorCode eventID, String eventMessage)	
回调说明	当发生 GizEventType 中列举的事件类型时，SDK 会主动触发该回调，该回调通知的主要是发生的异常事件	
回调参数	eventType	事件类型。指明发生了哪一类的事件，详细见 GizEventType 枚举定义
	eventSource	事件源，指是谁触发的事件。如果 eventType 是 GizEventSDK，eventSource 为 null；如果是 GizEventDevice，eventSource 需要强制转换为 GizWifiDevice 类型再使用；如果是 GizEventM2Mservice 或者 GizEventToken，eventSource 需要强制转换为 String 类型再使用
	eventID	事件 ID。代表事件编号，详细见 GizWifiErrorCode 枚举定义。该参数指出 eventSource 发生了什么事
	eventMessage	事件 ID 的消息描述

代码示例	// 设置 SDK 监听
	GizWifiSDK.sharedInstance().setListener(mListener);	

```
// 设置 AppInfo
ConcurrentHashMap<String, Object> appInfo = new ConcurrentHashMap<String,
String>();
appInfo.put("appId", "your_app_id");
appInfo.put("appSecret", "your_app_secret");

// 设置要过滤的设备 productKey 列表。不过滤则直接传 null
List<String> productInfo = new ArrayList<String> ();
ConcurrentHashMap<String, Object> product = new ConcurrentHashMap<String,
Object>();
product.put("productKey", "your_product_secret");
product.put("productSecret", "your_product_secret");
productInfo.add(product);

// 指定要切换的域名信息。使用机智云生产环境则传 null
// ConcurrentHashMap<String, Object> cloudServiceInfo = new
ConcurrentHashMap<String, Object>();
// cloudServiceInfo.put("openAPIInfo", "your_api_domain");

// 调用 SDK 的启动接口
GizWifiSDK.sharedInstance().startWithAppInfo(context, appInfo, productInfo, null,
false);
```

```
// 实现系统事件通知回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didNotifyEvent(GizEventType eventType, Object eventSource,
GizWifiErrorCode eventID, String eventMessage) {
        if (eventType == GizEventType.GizEventSDK) {
            // SDK发生异常的通知
            Log.i("GizWifiSDK", "SDK event happened: " + eventID + ", " +
eventMessage);
        } else if (eventType == GizEventType.GizEventDevice) {
            // 设备连接断开时可能产生的通知
            GizWifiDevice mDevice = (GizWifiDevice)eventSource;
            Log.i("GizWifiSDK", "device mac: " + mDevice.getMacAddress() + "
disconnect caused by eventID: " + eventID + ", eventMessage: " + eventMessage);
        } else if (eventType == GizEventType.GizEventM2MService) {
            // M2M服务返回的异常通知
            Log.i("GizWifiSDK", "M2M domain " + (String)eventSource + " exception
happened, eventID: " + eventID + ", eventMessage: " + eventMessage);
        } else if (eventType == GizEventType.GizEventToken) {
            // token失效通知
            Log.i("GizWifiSDK", "token " + (String)eventSource + " expired: " +
```

```
eventMessage);
            }
        }
};
```

【getCurrentCloudService】

定义	public void getCurrentCloudService()	
功能描述	查询当前使用的云服务域名信息	
回调	public void didGetCurrentCloudService(GizWifiErrorCode result, ConcurrentHashMap<String, String> cloudServiceInfo)	
回调说明	查询结果	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，cloudServiceInfo 为 null

	cloudServiceInfo	当前域名信息，字典{key: value}格式：
		{
		"openAPIDomain" : "xxx", // String类型
		"openAPIPort" : "xxx", // String类型
		"siteDomain" : "xxx", // String类型
		"sitePort" : "xxx", // String类型
		}

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);	

```
GizWifiSDK.sharedInstance().getCurrentCloudService();


// 实现回调
public void didGetCurrentCloudService(GizWifiErrorCode result,
ConcurrentHashMap<String, String> cloudServiceInfo) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 成功
    } else {
        // 失败
    }
}
```

【getVersion】

定义	public String getVersion()
功能描述	获取 SDK 的版本号。
返回值	SDK 的版本号
代码示例	String sdkVersion = GizWifiSDK.sharedInstance().getVersion();

【setLogLevel】

定义	public void setLogLevel(GizLogPrintLevel logLevel)
功能描述	设置日志输出级别。该级别指日志在调试终端的输出级别，默认是全部输出的。

	日志输出级别不影响日志文件的输出，无论日志输出级别设成什么，SDK 都会将运行日志写入文件。日志文件存放在 SD 卡目录下：GizWifiSDK/包名/GizSDKLog/	
参数	logLevel	日志输出级别，参考 GizLogPrintLevel 定义
代码示例	GizWifiSDK.sharedInstance().setLogLevel(GizLogPrintLevel. GizLogPrintAll);	

【disableLAN】

定义	public void disableLAN(boolean disabled)	
功能描述	设置是否禁用小循环功能	
参数	disabled	禁用或启用小循环
回调	public void didDisableLAN(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().disableLAN();


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didDisableLAN(GizWifiErrorCode result) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 设置成功
```

```
        } else {
            // 设置失败
        }
    }
};
```

【getSSIDList】

定义	public void getSSIDList()	
功能描述	在 Soft-AP 模式时，获得设备的 SSID 列表。SSID 列表通过异步回调方式返回	
回调	public void didGetSSIDList(GizWifiErrorCode result, List<GizWifiSSID> ssidInfoList)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，ssidInfoList 为 null
	ssidInfoList	由 GizWifiSSID 实例组成的 SSID 信号列表

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().getSSIDList();
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didGetSSIDList(GizWifiErrorCode result, List<GizWifiSSID>
	ssidInfoList) {	

```
            if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
                // 获取成功
            } else {
                // 获取失败
            }
        }
    };
```

【setDeviceOnboarding】

定义	public void setDeviceOnboarding(String ssid, String key, GizWifiConfigureMode mode, String softAPSSIDPrefix, int timeout, List<GizWifiGAgentType> types)	
功能描述	把设备配置到局域网 wifi 上。 设备处于 softap 模式时，模组会产生一个热点名称，手机 wifi 连接此热点后就可以配置了。 如果是机智云提供的固件，模组热点名称前缀为"XPG-GAgent-"，密码为"123456789"。 设备处于 airlink 模式时，手机随时都可以开始配置。但无论哪种配置方式，设备上线时，手机要连接到配置的局域网 wifi 上，才能够确认设备已配置成功。 设备配置成功时，在回调中会返回设备 mac 地址。如果设备重置了，设备 did 可能要在设备搜索回调中才能获取。	
参数	ssid	要配置的路由 SSID
	key	要配置的路由密码
	mode	配置模式，详细见 GizWifiConfigureMode 枚举定义

	softAPSSIDPrefix	SoftAPMode 模式下 SoftAP 的 SSID 前缀或全名。默认前缀为：XPG-GAgent-，SDK 以此判断手机当前是否连上了设备的 SoftAP 热点。AirLink 配置时该参数无意义，传 null 即可
	timeout	配置的超时时间。SDK 默认执行的超时时间为 30 秒
	types	待配置的模组类型，是一个 GizWifiGAgentType 枚举数组。若不指定则默认配置乐鑫模组。GizWifiGAgentType 定义了 SDK 支持的所有模组类型。GizWifiGAgentType 还定义了一个 GizGAgentOther 枚举值，用于开发者使用自己的配置库进行设备配置
回调	public void didSetDeviceOnboarding(GizWifiErrorCode result, String mac, String did, String productKey)	
回调说明	注意：如果调用 startWithAppInfo 接口时指定了待筛选的 productInfo 集合，如果设备被成功配置到路由上了，会返回配置成功，但不会出现在设备列表中。	
回调参数	result	详细见 GizWifiErrorCode 枚举定义，GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他参数为 null
	mac	设备 mac 地址
	did	设备 did。配置成功时，did 的值可能为 null，因为设备刚配置完不一定会马上从云端申请到 did
	productKey	设备的产品类型标识

代码示例	// airlink 配置
	GizWifiSDK.sharedInstance().setListener(mListener);
	List<GizWifiGAgentType> types = new ArrayList();	

```
types.add(GizWifiGAgentType.GizGAgentESP);
GizWifiSDK.sharedInstance().setDeviceOnboarding("your_ssid", "your_key",
GizWifiConfigureMode.GizWifiAirLink, null, 60, types);

// softap 配置
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().setDeviceOnboarding("your_ssid", "your_key",
GizWifiConfigureMode.GizWifiSoftAP, "XPG-GAgent-DF4A", 60, null);

// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public  void didSetDeviceOnboarding(GizWifiErrorCode result, String mac,
    String did, String productKey) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 配置成功
        } else {
            // 配置失败
        }
    }
};
```

【setDeviceOnboardingByBind】

定义	public void setDeviceOnboardingByBind(String ssid, String key, GizWifiConfigureMode mode, String softAPSSIDPrefix, int timeout, List<GizWifiGAgentType> types)	
功能描述	把设备配置到局域网 wifi 上，配网成功时自动绑定设备。此接口要在用户登录成功后再调用。设备处于 softap 模式时，模组会产生一个热点名称，手机 wifi 连接此热点后就可以配置了。如果是机智云提供的固件，模组热点名称前缀为"XPG-GAgent-"，密码为"123456789"或无密码。设备处于 airlink 模式时，手机随时都可以开始配置。但无论哪种配置方式，设备上线时，手机要先连接到配置的局域网 wifi 上然后才能被绑定到用户账号下	
参数	ssid	要配置的路由 SSID
	key	要配置的路由密码
	mode	配置模式，详细见 GizWifiConfigureMode 枚举定义
	softAPSSIDPrefix	SoftAPMode 模式下 SoftAP 的 SSID 前缀或全名。默认前缀为：XPG-GAgent-，SDK 以此判断手机当前是否连上了设备的 SoftAP 热点。AirLink 配置时该参数无意义，传 null 即可
	timeout	配置的超时时间。SDK 默认执行的超时时间为 30 秒。在超时时间内如果无法配置和绑定会回调配网失败
	types	待配置的模组类型，是一个 GizWifiGAgentType 枚举数组。若不指定则默认配置乐鑫模组。GizWifiGAgentType 定义了 SDK 支持的所有模组类型。GizWifiGAgentType 还定义了一个 GizGAgentOther 枚举值，用于开发者使用自己的配置库进行设备配置
回调	public void didSetDeviceOnboarding(GizWifiErrorCode result, String mac, String did, String productKey)	
回调说明	注意：如果调用 startWithAppInfo 接口时指定了待筛选的 productInfo 集合，如果设备被	

	成功配置到路由上并绑定成功，会返回配置成功，但不会出现在设备列表中。	
回调参数	result	详细见 GizWifiErrorCode 枚举定义，GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他参数为 null
	mac	设备 mac 地址
	did	设备 did。配置成功时，did 的值可能为 null，因为设备刚配置完不一定会马上从云端申请到 did
	productKey	设备的产品类型标识

代码示例	// airlink 配置
	GizWifiSDK.sharedInstance().setListener(mListener);
	List<GizWifiGAgentType> types = new ArrayList();
	types.add(GizWifiGAgentType.GizGAgentESP);
	GizWifiSDK.sharedInstance().setDeviceOnboardingByBind("your_ssid", "your_key",
	GizWifiConfigureMode.GizWifiAirLink, null, 60, types);
	
	// softap 配置
	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().setDeviceOnboardingByBind("your_ssid", "your_key",
	GizWifiConfigureMode.GizWifiSoftAP, "XPG-GAgent-DF4A", 60, null);
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {	

```
    @Override
    public  void didSetDeviceOnboarding(GizWifiErrorCode result, String mac,
    String did, String productKey) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 配置成功
        } else {
            // 配置失败
        }
    }
};
```

【getDevicesToSetServerInfo】

定义	public void getDevicesToSetServerInfo()	
功能描述	获取可以设置域名的设备列表。该接口返回支持域名设置功能的设备信息列表，App 可以在给设备设置域名前，先调用该接口查看有哪些设备可以设置域名。	
回调	public void didGetDevicesToSetServerInfo(GizWifiErrorCode result, List<ConcurrentHashMap<String, String>> devices);	
回调说明	该回调接口只返回设备的 mac、productKey、domain 这三个信息，不返回设备对象	
回调参数	result	获取成功或失败。如果获取失败，其他参数为 null
	devices	设备信息列表。设备信息字典格式如下： {

		“mac”: “xxx” // 设备 mac 地址 “productKey”: “xxx” // 设备的 productKey “domain”: “xxx” // 设备的域名信息 }

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	
	// 获取可设置域名的设备列表
	GizWifiSDK.sharedInstance().getDevicesToSetServerInfo();
	
	// 实现回调
	public void didGetDevicesToSetServerInfo(GizWifiErrorCode result, List<ConcurrentHashMap<String, String>> devices) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 获取成功
	} else {
	// 获取失败
	}
	}	

【setDeviceServerInfo】

定义	public void setDeviceServerInfo(String domain, String mac)
功能描述	此接口为手动设置设备域名接口，可为设备设置对应的云服务域名。

	设备和手机都连接到同一个 wifi 路由器后，可以设置设备要连接的云服务域名。可以设置当前已上线的所有小循环设备的域名。也可以单独设置某个设备的域名。如果不知道设备的 MAC 地址，可以先调用 getDevicesToSetServerInfo 接口查看有哪些设备可以设置域名，再调用该接口进行设置。 注意： 1、只支持可设置域名的设备 2、调用该接口将关闭已开启的设备域名自动设置功能	
参数	domain	待设置的域名。若该参数为 null，SDK 将根据用户手机的地理位置信息为设备设置机智云统一部署的云服务域名。 若要让设备连接独立部署的私有云域名，该参数为对应的私有云域名字符串，格式为：api.xxxxxx.com。这里需保证传入的域名是有效的，否则可能导致设备无法正常工作
	mac	待设置的设备 mac。默认参数为 null，即所有已发现的小循环设备都会被修改域名。如果只设置特定设备的域名，需指定 mac 地址
回调	public void didSetDeviceServerInfo(GizWifiErrorCode result, String mac);	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	mac	设置域名的设备 mac

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	
	// 给设备设置域名
	GizWifiSDK.sharedInstance().setDeviceServerInfo(null, "your_device_mac");	

```
// 实现回调
public void didSetDeviceServerInfo(GizWifiErrorCode result, String mac) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 设置成功
    } else {
        // 设置失败
    }
}
```

【getBoundDevices】

定义	public void getBoundDevices(String uid, String token)	
功能描述	获取绑定设备列表。在不同的网络环境下，有不同的处理： 当手机能访问外网时，该接口会向云端发起获取绑定设备列表请求；当手机不能访问外网时，局域网设备是实时发现的，但会保留之前已经获取过的绑定设备；手机处于无网模式时，局域网未绑定设备会消失，但会保留之前已经获取过的绑定设备。用户未登录时，无法获取到绑定设备列表。 请注意：此接口传入的 uid、token，如果长度错误，SDK 会继续使用之前的 uid、token 作处理	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
回调	public void didDiscovered(GizWifiErrorCode result, List<GizWifiDevice> deviceList)	
回调说明	以下触发场景触发回调： getBoundDevices 接口调用时触发该回调，错误码代表云端请求状态，设备列表是绑定设备与	

<table>
<tr><td></td><td colspan="2">局域网设备合并之后的集合；
设备列表发生变化时会主动上报时触发该回调，此时错误码 GIZ_SDK_SUCCESS，设备列表仍然是合并过的集合。</td></tr>
<tr><td rowspan="2">回调参数</td><td>result</td><td>详细见 GizWifiErrorCode 枚举定义，GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，deviceList 为非 null 集合</td></tr>
<tr><td>deviceList</td><td>GizWifiDevice 实例组成的数组，该参数将只返回根据指定 productKey 筛选过的设备集合。productKey 在 getBoundDevices 接口调用时指定</td></tr>
<tr><td>代码示例</td><td colspan="2">

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().getBoundDevices("your_uid", "your_token");


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didDiscovered(GizWifiErrorCode result, List<GizWifiDevice>
    deviceList) {
        // 提示错误原因
        if(result != GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            Log.d("", "result: " + result.name());
        }
        // 显示设备列表
        Log.d("", "discovered deviceList: " + deviceList);
    }
```

</td></tr>
</table>

	};

【bindRemoteDevice】

定义	public void bindRemoteDevice(String uid, String token, String mac, String productKey, String productSecret)	
功能描述	绑定远端设备到服务器	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
	mac	待绑定设备的 mac
	productKey	待绑定设备的 productKey
	productSecret	待绑定设备的 productSecret
回调	public void didBindDevice(GizWifiErrorCode result, String did)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，did 为 null
	did	绑定成功的设备 did
示例	GizWifiSDK.sharedInstance().setListener(mListener); GizWifiSDK.sharedInstance().bindRemoteDevice ("your_uid", "your_token", "your_device_mac", "your_device_product_key", "your_product_secret"); // 实现回调	

```
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didBindDevice(GizWifiErrorCode result, String did) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 绑定成功
        } else {
            // 绑定失败
        }
    }
};
```

【unbindDevice】

定义	public void unbindDevice(String uid, String token, String did)	
功能描述	把设备从服务器解绑	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
	did	待解绑设备的 did
回调	public void didUnbindDevice(GizWifiErrorCode result, String did)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，did 为 null
	did	已解绑的设备 did

示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().unbindDevice("your_uid", "your_token", "your_device_did");
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didUnbindDevice(GizWifiErrorCode result, String did) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 解绑成功
	} else {
	// 解绑失败
	}
	}
	};

【getCaptchaCode】

定义	public void getCaptchaCode(String appSecret)	
功能描述	获取图片验证码。开发者登录 site.gizwits.com，在自己账户下的应用管理中可以得到 App Secret，通过应用的 App Secret 才能获取到图片验证码。	
参数	appSecret	应用的 secret 信息，从 site.gizwits.com 中可以看到

<table>
<tr><td>回调</td><td colspan="2">public void didGetCaptchaCode(GizWifiErrorCode result, String token, String captchaId, String captchaURL)</td></tr>
<tr><td rowspan="4">回调参数</td><td>result</td><td>详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 null</td></tr>
<tr><td>token</td><td>图片验证码 token。图片验证码 token 在 1 小时后过期</td></tr>
<tr><td>captchaId</td><td>图片验证码 id。图片验证码 5 分钟后过期</td></tr>
<tr><td>captchaURL</td><td>图片验证码网址。图片验证码 url 在使用后过期</td></tr>
<tr><td>代码示例</td><td colspan="2">

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().getCaptchaCode("your_app_secret");


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didGetCaptchaCode(GizWifiErrorCode result, String token, String captchaId, String captchaURL) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 获取成功
        } else {
            // 获取失败
        }
    }
```

</td></tr>
</table>

	};

【requestSendPhoneSMSCode】

定义	public void requestSendPhoneSMSCode(String appSecret, String phone)	
功能描述	通过手机号请求短信验证码	
参数	appSecret	应用的 secret 信息，从 site.gizwits.com 中可以看到
	phone	手机号
回调	public void didRequestSendPhoneSMSCode(GizWifiErrorCode result, String token)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，token 为 null
	token	请求短信验证码时得到的 token

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().requestSendPhoneSMSCode ("your_app_secret", "your_phone_number");
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didRequestSendPhoneSMSCode(GizWifiErrorCode result, String token) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {	

```
            // 请求成功
        } else {
            // 请求失败
        }
    }
};
```

【requestSendPhoneSMSCode】

定义	public void requestSendPhoneSMSCode(String token, String captchaId, String captchaCode, String phone)	
功能描述	通过图形验证码请求短信验证码	
参数	token	通过 getCaptchaCode 获取到的 token
	captchaId	通过 getCaptchaCode 获取到的 captchaId
	captchaCode	图片验证码的内容
	phone	手机号
回调	public void didRequestSendPhoneSMSCode(GizWifiErrorCode result, String token)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，token 为 null
	token	请求短信验证码时得到的 token
代码示例	GizWifiSDK.sharedInstance().setListener(mListener);	

```
GizWifiSDK.sharedInstance().requestSendPhoneSMSCode ("your_token",
"your_captchaId", "your_captchaCode", "your_phone_number");


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didRequestSendPhoneSMSCode(GizWifiErrorCode result, String
    token) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 请求成功
        } else {
            // 请求失败
        }
    }
};
```

【verifyPhoneSMSCode】

定义	public void verifyPhoneSMSCode(String token, String phoneCode, String phone)	
功能描述	验证手机短信验证码。注意，验证短信验证码后，验证码就失效了，无法再用于手机号注册	
参数	token	验证码的 token，通过 getCaptchaCode 获取
	phoneCode	手机短信验证码

	phone	手机号码
回调	public void didVerifyPhoneSMSCode(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().verifyPhoneSMSCode ("your_token",
"your_verify_code", "your_phone_number");


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didVerifyPhoneSMSCode (GizWifiErrorCode result) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 验证成功
        } else {
            // 验证失败
        }
    }
};
```

【registerUser】

定义	public void registerUser(String username, String password, String code, GizUserAccountType accountType)	
功能描述	用户注册。需指定用户类型注册。手机用户的用户名是手机号，邮箱用户的用户名是邮箱、普通用户的用户名可以是普通用户名	
参数	username	注册用户名（可以是手机号、邮箱或普通用户名）
	password	注册密码
	code	手机短信验证码。短信验证码注册后就失效了，不能被再次使用
	accountType	用户类型，详细见 GizUserAccountType 枚举定义。注册手机号时，此参数指定为手机用户，注册邮箱时，此参数指定为邮箱用户，注册普通用户名时，此参数指定为普通用户
回调	public void didRegisterUser(GizWifiErrorCode result, String uid, String token)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，uid、token 为 null
	uid	注册成功后得到的 uid
	token	注册成功后得到的 token

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().registerUser ("your_phone_number", "your_password", "your_verify_code", GizUserAccountType.GizUserPhone);
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {	

```
    @Override
    public void didRegisterUser(GizWifiErrorCode result, String uid,  String token)
    {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 注册成功
        } else {
            // 注册失败
        }
    }
};
```

【userLoginAnonymous】

定义	public void userLoginAnonymous()	
功能描述	匿名登录。匿名方式登录，不需要注册用户账号。	
回调	public void didUserLogin(GizWifiErrorCode result, String uid, String token)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，uid、token 为 null
	uid	注册成功后得到的 uid
	token	注册成功后得到的 token
代码示例	GizWifiSDK.sharedInstance().setListener(mListener); GizWifiSDK.sharedInstance().userLoginAnonymous();	

```
// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public  void didUserLogin(GizWifiErrorCode result, String uid,  String token) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 登录成功
        } else {
            // 登录失败
        }
    }
};
```

【userLogin】

定义	public void userLogin(String username, String password)	
功能描述	用户登录。需使用注册成功的用户名、密码进行登录，可以是手机用户名、邮箱用户名或普通用户名	
参数	username	注册成功的用户名
	password	注册成功的用户密码
回调	public void didUserLogin(GizWifiErrorCode result, String uid, String token)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其

		他为失败。失败时，uid、token 为 null
	uid	登录成功后得到的 uid
	token	登录成功后得到的 token

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().userLogin("your_user_name", "your_password");
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didUserLogin(GizWifiErrorCode result, String uid, String token) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 登录成功
	} else {
	// 登录失败
	}
	}
	};	

【userLoginWithThirdAccount】

定义	public void loginWithThirdAccount(GizThirdAccountType thirdAccountType, String uid, String token, String tokenSecret)

<table>
<tr><td>功能描述</td><td colspan="2">第三方账号登录（第三方接口登录方式）</td></tr>
<tr><td rowspan="4">参数</td><td>thirdAccountType</td><td>第三方账号类型，详细见 GizThirdAccountType 枚举定义</td></tr>
<tr><td>uid</td><td>通过第三方平台 api 方式登录后得到的 uid</td></tr>
<tr><td>token</td><td>通过第三方平台 api 方式 登录后得到的 token</td></tr>
<tr><td>tokenSecret</td><td>推特账号登录时需要通过推特平台 api 方式得到此参数，其他第三方账号此参数可传 null</td></tr>
<tr><td>回调</td><td colspan="2">public void didUserLogin(GizWifiErrorCode result, String uid, String token)</td></tr>
<tr><td rowspan="3">回调参数</td><td>result</td><td>详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，uid、token 为 null</td></tr>
<tr><td>uid</td><td>登录成功后得到的 uid</td></tr>
<tr><td>token</td><td>登录成功后得到的 token</td></tr>
<tr><td>代码示例</td><td colspan="2">

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().loginWithThirdAccount(GizThirdAccountType.GizThirdSINA, "your_third_uid", "your_third_token", null);


// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didUserLogin(GizWifiErrorCode result, String uid, String token) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 登录成功
```

</td></tr>
</table>

```
        } else {
            // 登录失败
        }
    }
};
```

【changeUserPassword】

定义	public void changeUserPassword(String token, String oldPassword, String newPassword)	
功能描述	修改用户密码	
参数	token	用户登录或注册时得到的 token
	oldPassword	旧密码
	newPassword	新密码
回调	public void didChangeUserPassword(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	GizWifiSDK.sharedInstance().setListener(mListener); GizWifiSDK.sharedInstance().changeUserPassword("your_token", "your_old_password", "your_new_password"); // 实现回调	

```
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public  void didChangeUserPassword(GizWifiErrorCode result) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 修改成功
        } else {
            // 修改失败
        }
    }
};
```

【resetPassword】

定义	public void resetPassword(String username, String code, String newPassword, GizUserAccountType accountType)	
功能描述	重置密码。手机号重置密码时通过手机短信验证码重置，邮箱重置密码时需通过邮箱密码重置链接重置	
参数	username	待重置密码的手机号或邮箱
	code	重置手机用户密码时需要使用手机短信验证码（通过 requestSendPhoneSMSCode 方法获取）
	newPassword	新密码。邮箱重置密码时不需要填充密码，可指定为 null
	accountType	用户类型，详细见 GizThirdAccountType 枚举定义。待重置密码的用户名是手机号时，此参数指定为手机用户，待重置密码的用户名是邮箱时，此参数指定为邮箱用户

回调	public void didChangeUserPassword(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
GizWifiSDK.sharedInstance().setListener(mListener);
GizWifiSDK.sharedInstance().resetPassword("your_phone_number",
"your_verify_code", "your_new_password", GizUserAccountType.GizUserPhone);

// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public void didChangeUserPassword(GizWifiErrorCode result) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 修改成功
        } else {
            // 修改失败
        }
    }
};
```

【changeUserInfo】

定义	public void changeUserInfo(String token, String username, String code,

	GizUserAccountType accountType, GizUserInfo additionalInfo)	
功能描述	修改用户信息，包括用户名和个人信息。用户名只支持修改手机号或邮箱，并且手机号或邮箱必须是没有注册过的。该接口用于以下场景：只修改手机号、只修改邮箱、只修改普通用户的个人信息、同时修改手机号和补充信息、同时修改邮箱和补充信息。只修改个人信息时，accountType 可以指定为 GizUserNormal；修改手机号要指定为 GizUserPhone；修改邮箱要指定为 GizUserEmail	
参数	token	用户登录或注册时得到的 token
	username	待修改的手机号或邮箱
	code	修改手机号时要使用的手机短信验证码
	accountType	用户类型，详细见 GizThirdAccountType 枚举定义。修改手机号时，accountType 传 GizUserPhone；修改普通用户名时，accountType 传 GizUserEmail；只修改个人信息时，accountType 传 GizUserNormal；同时修改用户名和个人信息时，可根据待修改的是手机号还是邮箱来指定。
	additionalInfo	待修改的个人信息，详细见 GizUserInfo 类定义。如果只修改个人信息，需要指定 token，username、code 填 null
回调	public void didChangeUserInfo(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	GizWifiSDK.sharedInstance().setListener(mListener); GizWifiSDK.sharedInstance().changeUserInfo("your_token", "your_phone_number", "your_verify_code", GizUserAccountType.GizUserPhone, null);	

```
// 实现回调
GizWifiSDKListener mListener = new GizWifiSDKListener() {
    @Override
    public  void didChangeUserInfo(GizWifiErrorCode result) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 修改成功
        } else {
            // 修改失败
        }
    }
};
```

【getUserInfo】

定义	public void getUserInfo(String token)	
功能描述	获取用户信息	
参数	token	用户登录或注册时得到的 token
回调	public void didGetUserInfo(GizWifiErrorCode result, GizUserInfo userInfo)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	userInfo	用户信息，详细见 GizUserInfo 类

代码示例	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().getUserInfo ("your_token");
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didGetUserInfo(GizWifiErrorCode result, GizUserInfo userInfo) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 获取成功
	} else {
	// 获取失败
	}
	}
	};

【transAnonymousUser】

定义	public void transAnonymousUser(String token, String username, String password, String code, GizUserAccountType accountType)	
功能描述	匿名用户转换，可转换为手机用户或者普通用户。注意，待转换的帐号必须是还未注册过的	
参数	token	用户登录或注册时得到的 token
	username	待转换的普通账号或手机号

	password	转换后的帐号密码
	code	转换为手机用户时要使用的手机短信验证码
	accountType	指定待转换的用户类型，详细见 GizThirdAccountType 枚举定义。待转换的用户名是手机号时，此参数指定为 GizUserPhone，待转换用户名是普通账号时，此参数指定为 GizUserNormal
回调	public void didTransAnonymousUser(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例	// 匿名转手机用户
	GizWifiSDK.sharedInstance().setListener(mListener);
	GizWifiSDK.sharedInstance().transAnonymousUser("your_token", "your_phone_number", "your_password", "your_verify_code", GizUserAccountType.GizUserPhone);
	
	// 实现回调
	GizWifiSDKListener mListener = new GizWifiSDKListener() {
	@Override
	public void didTransAnonymousUser(GizWifiErrorCode result) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 转换成功
	} else {	

```
            // 转换失败
        }
    }
};
```

【userFeedback】

定义	public void userFeedback(String contactInfo, String feedbackInfo, boolean sendLog)	
功能描述	用户信息反馈接口。此接口无回调，调用后就会上传信息。目前信息上传后，需要联系机智云 FAE 查看	
参数	contactInfo	用户的联系方式。此参数为选填
	feedbackInfo	用户反馈的信息。此参数为选填
	sendLog	是否发送问题日志。如果前面两个参数都没填，则默认发送问题日志
代码示例	GizWifiSDK.sharedInstance().userFeedback("your_phone", "your_message", true);	

2. GizWifiDevice 类

2.1. 简介

机智云 Wi-Fi 的设备类。GizWifiDevice 类为 APP 开发者提供设备订阅、设备数据通知、设备实时状态通知，例如热水器的水温等功能。该设备实例是通过 GizWifiDevice 类分配出来的，不能自行创建。

2.2. 属性方法

【setListener】

定义	public void setListener(GizWifiDeviceListener Listener)	
功能描述	设置设备的监听器	
参数	listener	设备监听器

示例	// mDevice 是从设备列表中获取到的设备实体对象
	
	mDevice.setListener(new GizWifiDeviceListener() {});	

【getMacAddress】

定义	public String getMacAddress()
功能描述	获取设备的 Mac 地址。如果是 VIRTUAL:SITE，则是虚拟设备
返回值	返回设备的 Mac 地址
示例	// mDevice 是从设备列表中获取到的设备实体对象 String mac = mDevice.getMacAddress();

【getDid】

定义	public String getDid()
功能描述	设备云端身份标识 DID
返回值	返回设备的 did
示例	// mDevice 是从设备列表中获取到的设备实体对象

	String did = mDevice.getDid();

【getIPAddress】

定义	public String getIpAddress()
功能描述	获取设备的 ip 地址。大循环设备的 ip 地址为云端服务器域名
返回值	返回设备的 ip 地址
示例	// mDevice 是从设备列表中获取到的设备实体对象 String ip = mDevice.getIPAddress();

【getProductKey】

定义	public String getProductKey()
功能描述	获取设备的产品类型识别码
返回值	返回设备的产品类型识别码
示例	// mDevice 是从设备列表中获取到的设备实体对象 String productKey = mDevice.getProductKey();

【getProductName】

定义	public String getProductName()
功能描述	获取设备的产品名称
返回值	返回设备的产品名称
示例	// mDevice 是从设备列表中获取到的设备实体对象

	String productName = mDevice.getProductName();

【getProductType】

定义	public GizWifiDeviceType getProductType()
功能描述	获取设备分类是中控设备还是普通设备
返回值	返回设备分类
示例	// mDevice 是从设备列表中获取到的设备实体对象 GizWifiDeviceType type = mDevice.getProductType();

【getRemark】

定义	public String getRemark()
功能描述	获取设备的备注信息。设备绑定后可以修改，默认为空
返回值	返回设备的备注信息
示例	// mDevice 是从设备列表中获取到的设备实体对象 String remark = mDevice.getRemark();

【getAlias】

定义	public String getAlias()
功能描述	获取设备的别名。设备绑定后可以修改，默认为空
返回值	返回设备的别名
示例	// mDevice 是从设备列表中获取到的设备实体对象

	String alias = mDevice.getAlias();

【getNetStatus】

定义	public GizWifiDeviceNetStatus getNetStatus()
功能描述	获取设备的网络状态，详见 GizWifiDeviceNetStatus 枚举定义
返回值	返回设备的网络状态
示例	// mDevice 是从设备列表中获取到的设备实体对象 GizWifiDeviceNetStatus netStatus = mDevice. getNetStatus ();

【isLAN】

定义	public boolean isLAN()
功能描述	判断设备是小循环还是大循环
返回值	返回设备是小循环还是大循环
示例	// mDevice 是从设备列表中获取到的设备实体对象 boolean isBind = mDevice.isLAN();

【isBind】

定义	public boolean isBind()
功能描述	判断设备是否已绑定
返回值	返回设备是否已绑定
示例	// mDevice 是从设备列表中获取到的设备实体对象

	boolean isBind = mDevice.isBind();

【isDisabled】

定义	public boolean isDisabled()
功能描述	判断设备是否已在云端注销
返回值	返回设备是否已注销
示例	// mDevice 是从设备列表中获取到的设备实体对象 boolean isDisabled = mDevice.isDisabled();

【isSubscribed】

定义	public boolean isSubscribed()
功能描述	判断设备是否已订阅
返回值	返回设备是否已订阅
示例	// mDevice 是从设备列表中获取到的设备实体对象 boolean isSubscribed = mDevice.isSubscribed();

【isProductDefined】

定义	public boolean isProductDefined()
功能描述	判断设备是否定义了产品数据点
返回值	返回设备是否有数据点定义
示例	// mDevice 是从设备列表中获取到的设备实体对象

	boolean isProductDefined = mDevice.isProductDefined();

【getSharingRole】

定义	public GizDeviceSharingUserRole getSharingRole()
功能描述	获取绑定设备的用户权限，
返回值	返回绑定用户的权限。见 GizDeviceSharingUserRole 枚举定义
示例	// mDevice 是从设备列表中获取到的设备实体对象 GizDeviceSharingUserRole role = mDevice. getSharingRole();

2.3. 回调接口

以下是 GizWifiDevice 类提供的所有回调接口，将在在后续 API 定义中详细介绍：

- didGetHardwareInfo：设备硬件信息的回调
- didSetCustomInfo：设置设备绑定信息的回调
- didExitProductionTesting：设备退出产测的回调
- didSetSubscribe：设备订阅或解除订阅的回调
- didUpdateNetStatus：设备网络状态变化通知
- didReceiveData：接收到设备状态上报的回调

2.4. API

【didUpdateNetStatus】

回调	public void didUpdateNetStatus(GizWifiDevice device, GizWifiDeviceNetStatus netStatus)

回调说明	该回调主动上报设备的网络状态变化，当设备重上电、断电或可控时会触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	netStatus	设备是离线、在线还是可控状态

代码示例	// mDevice是从设备列表中获取到的设备实体对象
	mDevice.setListener(mListener);
	
	// 实现回调
	GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
	@override
	public void didUpdateNetStatus(GizWifiDevice device, GizWifiDeviceNetStatus
	netStatus) {
	}
	};	

【setSubscribe】

定义	public void setSubscribe(String productSecret, boolean subscribed)
功能描述	设备订阅或解除订阅。订阅了设备，表示使用者关心这个设备的消息推送。解除订阅，表示使用者不关心这个设备的消息推送。订阅设备后，SDK 将自动登录和自动绑定设备。解除订阅后，设备连接将自动断开，但不会自动解绑。一般来说，设备订阅都会成功的，SDK 会记住设备是否被订阅了。 若 startWithAppInfo 中传递了 productSecret 有效，此接口就忽略 productSecret 参数，否则使用此接口参数做订阅

参数	productSecret	设备的产品秘钥。在机智云开发者中心 dev.gizwits.com 的“产品信息”中，可以看到与 Product Key 对应的 Product Secret。此参数无默认值，开发者必须传入正确的 productSecret
	isSubscribed	订阅或取消订阅。true 表示订阅，false 表示取消订阅
回调	public void didSetSubscribe(GizWifiErrorCode result, GizWifiDevice device, boolean isSubscribed)	
回调参数	device	回调的 GizWifiDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，订阅状态无变化
	isSubscribed	设备是被订阅了还是被解除订阅了。true 表示被订阅，false 表示被解除订阅

代码示例	// mDevice 是从设备列表中获取到的设备实体对象
	mDevice.setListener(mListener);
	mDevice.setSubscribe(true); // 订阅设备
	mDevice.setSubscribe(false); // 解除订阅
	
	// 实现回调
	GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
	@Override
	public void didSetSubscribe(GizWifiErrorCode result, GizWifiDevice device, boolean isSubscribed) {	

```
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 订阅或解除订阅成功
    } else {
        // 失败
    }
}
};
```

【getDeviceStatus】

定义	public void getDeviceStatus(List<String> attrs)	
功能描述	获取设备状态。已订阅的设备变为可控状态后才能获取到状态。如果设备是变长数据点类型，则可查询指定的数据点状态	
参数	attrs	要查询状态的数据点名称，为 String 类型数组。此参数默认值为 null。SDK 默认返回设备的所有数据点状态。若要查询某些数据点的状态，参数应指定为要查询的数据点名称数组
回调	public void didReceiveData(GizWifiErrorCode result, GizWifiDevice device, ConcurrentHashMap<String, Object> dataMap, int sn)	
回调说明	设备回复或上报的数据中，当 SDK 遇到无法解析的数据时，会作为透传数据处理，此时错误码为 GIZ_SDK_SUCCESS	
回调参数	device	回复状态的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，dataMap 为空字典

	data	设备上报的数据内容，字典格式：
		{
		"data": [value]， // value 为 ConcurrentHashMap 类型，内容为设备状态键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致
		"alerts": [value]， // value 为 ConcurrentHashMap 类型，内容为设备报警键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致
		"faults": [value]， // value 为 ConcurrentHashMap 类型，内容为设备故障键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致
		"binary": [value]， // value 为 Byte[]类型，内容为二进制数据，指没有在 site 上定义数据点的需要透传的数据
		}

	sn	控制指令的应答序号，此应答序号与 APP 发送控制指令的序号一致。设备主动上报数据和回复状态查询时，序号为 0

代码示例		// mDevice 是从设备列表中获取到的设备实体对象
		mDevice.setListener(mListener);
		mDevice.getDeviceStatus(null);
		
		// 实现回调
		GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
		@Override
		public void didReceiveData(GizWifiErrorCode result, GizWifiDevice device, ConcurrentHashMap<String, Object> dataMap, int sn) {

```
            if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
                // 查询成功
            } else {
                // 查询失败
            }
        }
    };
```

【write】

定义	public void write(ConcurrentHashMap<String, Object> data, int sn)	
功能描述	给设备发送控制指令。已订阅的设备变为可控状态后才能发送控制指令	
参数	data	该参数为要发给设备的操作指令。为字典格式，字典键值对可按以下方式填充： 1、如果设备有数据点定义，操作指令一次可以下发多个数据点。字典中的 key 为数据点名称，value 为数据点的值。value 类型要与数据点定义一致： （1）如果数据点为布尔类型，则 value 为 boolean 类型； （2）如果数据点为数值类型，则 value 为 int 或 float 类型； （3）如果数据点为枚举类型，则 value 为枚举序号（int 类型）或者枚举字符串（String 类型）； （4）如果数据点为扩展类型，则 value 为 Byte[]类型； 2、如果设备操作采用透传方式，透传指令一次只能下发一条。字典中的 key 填充为”binary”，value 为 Byte[]类型。
	sn	控制指令序号，用于对应控制指令应答数据。控制确认回调时会返回这个 sn

回调	public void didReceiveData(GizWifiErrorCode result, GizWifiDevice device, ConcurrentHashMap<String, Object> dataMap, int sn)	
回调说明	设备回复或上报的数据中，当 SDK 遇到无法解析的数据时，会作为透传数据处理，此时错误码为 GIZ_SDK_SUCCESS	
回调参数	device	回复状态的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，dataMap 为空字典

	data	设备上报的数据内容，字典格式：
		{
		"data": [value], // value为ConcurrentHashMap类型，内容为设备状态键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致
		"alerts": [value], // value为ConcurrentHashMap类型，内容为设备报警键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致
		"faults": [value], // value为ConcurrentHashMap类型，内容为设备故障键值对，[数据点标识名：数据点值]，数据点值的类型与site上的定义一致
		"binary": [value], // value为Byte[]类型，内容为二进制数据，指没有在site上定义数据点的需要透传的数据
		}

	sn	控制指令的应答序号，此应答序号与 APP 发送控制指令的序号一致。设备主动上报数据和回复状态查询时，序号为0
代码示例	// mDevice 是从设备列表中获取到的设备实体对象，设置监听	

```
mDevice.setListener(mListener);


/*
 * 此代码为使用 sn 的示例。如果 App 使用命令序号 sn，sn 可设为相应的值
 */
// 订阅设备并变为可控状态后，执行开灯动作
int sn = 0;
ConcurrentHashMap command = new ConcurrentHashMap<String, boolean> ();
command.put("LED_OnOff", true);
mDevice.write(command, sn);


// 实现回调
GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
@Override
public void didReceiveData(GizWifiErrorCode result, GizWifiDevice device,
ConcurrentHashMap<String, Object> dataMap, int sn) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 开灯成功
        } else {
            // 开灯失败
        }
    }
```

	};

【setCustomInfo】

定义	public void setCustomInfo(String remark, String alias)	
功能描述	修改设备的备注和别名。设备绑定后才能修改	
参数	remark	待修改的备注信息。传 null 表示不修改，传""则会覆盖为空串
	alias	待修改的设备别名。传 null 表示不修改，传""则会覆盖为空串
回调	public void didSetCustomInfo(GizWifiErrorCode result, GizWifiDevice device)	
回调参数	device	修改备注和别名的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
// mDevice是从设备列表中获取到的设备实体对象
mDevice.setListener(mListener);
mDevice.setCustomInfo("your_remark", "your_alias");


// 实现回调
GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
    @Override
    public void didSetCustomInfo(GizWifiErrorCode result, GizWifiDevice device) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 修改成功
```

```
        } else {
            // 修改失败
        }
    }
};
```

【getHardwareInfo】

定义	public void getHardwareInfo()	
功能描述	获取硬件信息。不订阅设备也可以使用此接口，只要设备连入正常工作模式即可	
回调	public void didGetHardwareInfo(GizWifiErrorCode result, GizWifiDevice device, ConcurrentHashMap<String, String> hardwareInfo)	
回调参数	device	返回硬件信息的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，hardwareInfo 为 null
	hardwareInfo	硬件信息。对应的硬件信息键值对有： { "wifiHardVersion": [value], // value为String 类型，设备的 Wifi 模组硬件版本号 "wifiSoftVersion": [value], // value为String 类型，设备的 Wifi 模组软件版本号 "wifiFirmwareId": [value], // value为String 类型，设备的 Wifi

		固件 ID "wifiFirmwareVer": [value], // value为String 类型，设备的 Wifi 固件版本 "mcuHardVersion": [value], // value为String 类型，设备的硬件版本号 "mcuSoftVersion": [value], // value为String 类型，设备的软件版本号 "productKey": [value], // value为String 类型，设备的产品唯一标识码 }

代码示例		// mDevice是从设备列表中获取到的设备实体对象
		mDevice.setListener(mListener);
		mDevice.getHardwareInfo();
		
		// 实现回调
		GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
		@Override
		public void didGetHardwareInfo(GizWifiErrorCode result, GizWifiDevice device,
		ConcurrentHashMap<String, String> hardwareInfo) {
		if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
		// 获取成功
		} else {
		// 获取失败

```
        }
    }
};
```

【exitProductionTesting】

定义	public void exitProductionTesting()	
功能描述	让设备退出产测模式。不订阅设备就可以调用此接口，设备进入产测模式后会响应	
回调	public void didExitProductionTesting (GizWifiErrorCode result, GizWifiDevice device)	
回调参数	device	返回硬件信息的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
// mDevice是从设备列表中获取到的设备实体对象
mDevice.setListener(mListener);
mDevice.exitProductionTesting();


// 实现回调
GizWifiDeviceListener mListener = new GizWifiDeviceListener() {
    @Override
    public void didExitProductionTesting(GizWifiErrorCode result, GizWifiDevice
    device) {
```

```
            if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
                // 执行成功
            } else {
                // 执行失败
            }
        }
    };
```

3. GizWifiCentralControlDevice 类

3.1. 简介

GizWifiCentralControlDevice 类为 APP 开发者提供中控子设备操作，包括获取子设备列表、添加子设备、删除子设备等功能。

该类继承自 GizWifiDevice 类，除下列属性和方法外，也具备 GizWifiDevice 类的所有属性和方法。

3.2. 属性

属性	描述
subDeviceList	List<GizWifiDevice>类型，提供 get 方法。中控子设备列表

3.3. 回调接口

以下是 GizWifiCentralControlDevice 类提供的所有回调接口：

- didUpdateSubDevices：中控子设备列表回调

【didUpdateSubDevices】

定义	public void didUpdateSubDevices(GizWifiCentralControlDevice device, GizWifiErrorCode result, List<GizWifiDevice> subDeviceList)	
功能描述	子设备列表回调接口。添加、删除、同步更新子设备列表以及子设备列表变化上报都使用该回调接口	
回调参数	device	触发回调的 GizWifiCentralControlDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表；其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	subDeviceList	子设备列表。GizWifiDevice 对象数组

代码示例	// mDevice为从设备列表中取到的中控设备对象
	mDevice.setListener(mListener);
	
	GizWifiDeviceListener mListener = new GizWifiCentralControlDeviceListener() {
	@Override
	public void didUpdateSubDevices(GizWifiCentralControlDevice device, GizWifiErrorCode result, List<GizWifiDevice> subDeviceList) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	//接收变更的子设备列表	

```
        } else {
            //失败处理
        }
    }
};
```

3.4. API

【addSubDevice】

定义	public void addSubDevice(List<String> deviceMacs)	
功能描述	添加子设备。该接口让中控处于组网模式，等待子设备入网。只有中控设备可控后才能执行此操作。该接口会向中控设备发送添加子设备请求，中控设备将添加后的子设备列表通过回调返回	
参数	deviceMacs	要添加的子设备 mac 地址数组。默认为 null，默认时中控添加所有能够加入中控的子设备。如果指定 mac 地址则中控只添加这些特定的子设备

代码示例	// mDevice是从设备列表中获取到的中控设备实体对象
	mDevice.setListener(mListener);
	mDevice.addSubDevice(null);
	
	// 实现回调
	GizWifiDeviceListener mListener = new
	GizWifiCentralControlDeviceListener() {
	@Override	

```
    public void didUpdateSubDevices(GizWifiCentralControlDevice device,
    GizWifiErrorCode result, List<GizWifiDevice> subDeviceList) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 添加成功
        } else {
            // 添加失败
        }
    }
};
```

【deleteSubDevice】

定义	public void deleteSubDevice(GizWifiDevice device)	
功能描述	删除子设备，只有中控设备可控后才能执行此操作。该接口会向中控设备发送删除子设备请求，中控设备将删除后的子设备列表通过回调返回	
参数	device	待删除的子设备对象。在中控设备的子设备列表中找到子设备，设备对象传入该参数。此参数不能为 null

代码示例	// mDevice是从设备列表中获取到的中控设备实体对象
	mDevice.setListener(mListener);
	// mSubDevice是从子设备列表中获取到的要删除的设备实体对象
	mDevice.deleteSubDevice(mSubDevice);
	
	// 实现回调	

```
GizWifiDeviceListener mListener = new
GizWifiCentralControlDeviceListener() {
    @Override
    public void didUpdateSubDevices(GizWifiCentralControlDevice device,
    GizWifiErrorCode result, List<GizWifiDevice> subDeviceList)  {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 删除成功
        } else {
            // 删除失败
        }
    }
};
```

【updateSubDevices】

定义	public void updateSubDevices()
功能描述	同步更新子设备列表。只有中控设备可控后才能执行此操作。该接口会向中控设备发送获取子设备列表请求，中控设备将子设备列表通过回调返回

代码示例	// mDevice是从设备列表中获取到的中控设备实体对象
	mDevice.setListener(mListener);
	mDevice.updateSubDevices();
	
	// 实现回调

```
GizWifiDeviceListener mListener = new
GizWifiCentralControlDeviceListener() {
    @Override
    public void didUpdateSubDevices(GizWifiCentralControlDevice device,
    GizWifiErrorCode result, List<GizWifiDevice> subDeviceList) {
        if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
            // 更新成功
        } else {
            // 更新失败
        }
    }
};
```

4. GizUserInfo 类

4.1. 简介

GizUserInfo 类为开发者提供用户信息修改和获取。

4.2. 属性

属性	描述
uid	String 类型。用户登录后得到的 uid，提供 get 方法
username	String 类型。用户名：手机号或者邮箱，提供 get 方法

属性	描述
email	String 类型。用户邮箱，提供 get 方法
phone	String 类型。用户手机号，提供 get 方法
isAnonymous	boolean 类型。是否为匿名用户，提供 get 方法
lang	String 类型。用户的语言环境，提供 get 方法
name	String 类型。用户昵称，提供 get、set 方法
userGender	GizUserGenderType 类型。用户性别，提供 get、set 方法
birthday	String 类型。用户生日，提供 get、set 方法
address	String 类型。用户家庭住址，提供 get、set 方法
remark	String 类型。用户的备注信息，提供 get、set 方法
deviceBindTime	String 类型。此变量表示用户绑定设备的时间

5. GizWifiSSID 类

5.1. 简介

路由的 SSID 信息类，包括 Wifi 信号名称 SSID 和信号强度。

5.2. 属性

属性	描述
ssid	SSID 名。我们连接一个 Wi-Fi 热点时，可以搜索到的名字
rssi	热点对应的信号强度。取值范围 0-100

【getSsid】

定义	public String getSsid()
功能描述	获取 wifi 的 SSID 名称。我们连接一个 Wi-Fi 热点时，可以搜索到的名字
返回值	Wifi 的 ssid 名称
代码示例	// mWifiSSID 是 SDK 提供的热点列表中的 ssid 实体对象 String ssid = mWifiSSID.getSsid();

【getRssi】

定义	public int getRssi()
功能描述	热点对应的信号强度。取值范围 0-100
返回值	Wifi 的 ssid 名称
代码示例	// mWifiSSID 是 SDK 提供的热点列表中的 ssid 实体对象 int rssi = mWifiSSID.getRssi();

6. GizDeviceSharing 类

6.1. 简介

GizDeviceSharing 类为 APP 开发者提供设备分享功能，用户绑定设备后，其他人可以通过设备分享的方式使用设备。与设备分享的有关的用户分为四类：normal、specail、owner、guest，下面简单介绍这几类用户的权限：

normal：设备没有被分享过时，任何已绑定的用户都是 normal 用户，设备仍然可以被其他用户绑定；

special：只有第一个绑定设备的用户才可以分享设备，并成为 owner

owner：用户有 owner 后，其他用户不可以再绑定设备，只能通过分享的方式使用设备。owner 用户可以解绑所有其他已绑定用户

guest：接受分享邀请的用户是 guest 用户

6.2. 回调接口

以下是 GizDeviceSharing 类提供的所有回调接口，将在后续 API 定义中详细介绍：

- didGetBindingUsers：获取设备已绑定用户的回调
- didUnbindUser：解绑设备已绑用户的回调
- didGetDeviceSharingInfos：获取分享邀请列表的回调
- didSharingDevice：创建分享邀请的回调
- didRevokeDeviceSharing：撤回分享邀请的回调
- didAcceptDeviceSharing：接受分享邀请的回调
- didCheckDeviceSharingInfoByQRCode：查看二维码邀请信息的回调
- didAcceptDeviceSharingByQRCode：扫码接受分享邀请的回调
- didModifySharingInfo：修改分享别名的回调
- didQueryMessageList：查询消息列表的回调
- didMarkMessageStatus：标记或删除消息的回调

6.3. API

【setListener】

定义	public static void setListener(GizDeviceSharingListener listener)

功能描述	设置设备分享监听	
参数	listener	设备分享的监听
代码示例	GizDeviceSharing.setListener(mListener);	

【getBindingUsers】

定义	public static void getBindingUsers(String token, String deviceID)	
功能描述	查询设备的已绑定用户列表。只有 owner 用户才能查询设备的已绑用户	
参数	token	用户 token
	deviceID	要查询的设备 did
回调	public void didGetBindingUsers(GizWifiErrorCode result, String deviceID, List<GizUserInfo> bindUsers)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，bindUsers 回调参数为 null
	deviceID	发起查询的设备 ID
	bindUsers	String 类型数组，设备的已绑定用户列表。失败时为 null

代码示例	// 查询设备的已绑定用户列表
	GizDeviceSharing.setListener(mListener);
	GizDeviceSharing.getBindingUsers("your_token", "your_device_id");
	
	// 实现回调	

```
GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didGetBindingUsers(GizWifiErrorCode result, String deviceID,
List<GizUserInfo> bindUsers) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 查询成功
} else {
// 查询失败
}
}
};
```

【unbindUser】

定义	public static void unbindUser(String token, String deviceID, String guestUID)	
功能描述	解绑设备的已绑定用户。只有 owner 才能解绑其他已绑用户	
参数	token	用户 token
	deviceID	要解绑用户的设备 ID
	guestUID	要解绑的用户 ID
回调	public void didUnbindUser(GizWifiErrorCode result, String deviceID, String guestUID)	

回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	解绑用户的设备 ID
	guestUID	解绑的用户 ID

代码示例

```
GizDeviceSharing.setListener(mListener);


// 解绑其他用户
GizDeviceSharing.unbindUser("your_token", "your_device_id", "your_guest_uid");


// 实现回调
GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didGetBindingUsers(GizWifiErrorCode result, String deviceID,
List<GizUserInfo> bindUsers) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 解绑成功
} else {
// 解绑失败
}
}
};
```

【getDeviceSharingInfos】

定义	public static void getDeviceSharingInfos(String token, GizDeviceSharingType sharingType, String deviceID)	
功能描述	查询设备的分享邀请列表。可以查询自己发起的分享邀请，或者查询分享给自己的分享邀请，owner 和 guest 用户都可以查询	
参数	token	用户 token
	sharingType	要查询的分享邀请类型是分享给自己的还是自己分享给别人的，见枚举定义 GizDeviceSharingType
	deviceID	查询分享邀请的设备 ID。如果是 guest 用户，可查询所有邀请。如果是 owner 用户，可以指定设备 ID 查询，也可以不指定设备 ID 查询
回调	public void didGetDeviceSharingInfos(GizWifiErrorCode result, String deviceID, List<GizDeviceSharingInfo> deviceSharingInfos)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	查询分享邀请的设备 ID
	deviceSharingInfos	GizDeviceSharingInfo 类对象数组，分享邀请列表。如果失败，此参数为 null
代码示例	GizDeviceSharing.setListener(mListener); // 查询自己发出的分享邀请	

```
GizDeviceSharing.getDeviceSharingInfos("your_token",
GizDeviceSharingType.GizDeviceSharingByMe, "your_device_id");


// 实现回调
GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didGetDeviceSharingInfos(GizWifiErrorCode result, String deviceID,
List<GizDeviceSharingInfo> deviceSharingInfos) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 查询成功
} else {
// 查询失败
}
}
};
```

【sharingDevice】

定义	public static void sharingDevice(String token, String deviceID, GizDeviceSharingWay sharingWay, String guestUser, GizUserAccountType guestUserType)
功能描述	创建分享邀请。special 和 owner 用户可以通过账号分享或二维码分享的方式分享设备。账号

	分享邀请 24 小时后失效，二维码邀请 15 分钟后失效	
参数	token	用户 token
	deviceID	创建分享邀请的设备 ID
	sharingWay	分享邀请是通过账号分享还是二维码分享，见 GizDeviceSharingWay 枚举定义
	guestUser	如果是账号分享，要指定用户名，用户名可以是普通用户名、手机号、邮箱、用户的 uid。如果是二维码分享，该参数可传 null
	guestUserType	账号分享时，该参数需要指定用户名是哪种类型，见 GizUserAccountType 枚举定义。如果是通过用户的 uid 分享的，此变量应为 GizUserOther，其他按照对应的用户类型传值
回调	public void didSharingDevice(GizWifiErrorCode result, String deviceID, int sharingID, Bitmap QRCodeImage)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	创建分享邀请的设备 ID
	sharingID	分享邀请创建成功时被分配的 ID。失败时该参数为 null
	QRCodeImage	二维码图片内容。二维码邀请创建失败或者账号分享时，该参数为 null
代码示例	GizDeviceSharing.setListener(mListener); // 通过手机号分享设备 GizDeviceSharing.getDeviceSharingInfos("your_token", "your_device_id",	

```
GizDeviceSharingWay.GizDeviceSharingByNormal, "guest_phone_number",
GizUserAccountType.GizUserPhone);


GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didSharingDevice(GizWifiErrorCode result, String deviceID, int
sharingID, Bitmap QRCodeImage) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 分享成功
} else {
// 分享失败
}
}
};
```

【revokeDeviceSharing】

定义	public static void revokeDeviceSharing(String token, int sharingID)	
功能描述	撤回分享邀请。只有 owner 才能撤回自己的分享邀请，已经发出的分享邀请，可以随时撤回。一旦撤回成功，guest 用户会被解绑不能使用该设备	
参数	token	用户 token
	sharingID	要撤回的分享邀请 ID

<table>
<tr><td>回调</td><td colspan="2">public void didRevokeDeviceSharing(GizWifiErrorCode result, int sharingID)</td></tr>
<tr><td rowspan="2">回调参数</td><td>result</td><td>详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败</td></tr>
<tr><td>sharingID</td><td>撤回的分享邀请 ID</td></tr>
<tr><td>代码示例</td><td colspan="2">GizDeviceSharing.setListener(mListener);

// 撤回分享邀请
GizDeviceSharing.revokeDeviceSharing("your_token", your_sharing_id);

GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didRevokeDeviceSharing(GizWifiErrorCode result, int sharingID) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 撤回成功
} else {
// 撤回失败
}
}
};</td></tr>
</table>

【acceptDeviceSharing】

定义	public static void acceptDeviceSharing(String token, int sharingID, boolean accept)	
功能描述	接受或拒绝分享邀请。owner 用户以账号方式分享设备后，guest 账号可以接受或拒绝邀请	
参数	token	用户 token
	sharingID	要接受的分享邀请 ID
	accept	接受或拒绝邀请。true 表示接受，false 表示拒绝
回调	public void didAcceptDeviceSharing(GizWifiErrorCode result, int sharingID)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sharingID	接受或拒绝的邀请 ID

代码示例

```
GizDeviceSharing.setListener(mListener);


// 接受邀请
GizDeviceSharing.acceptDeviceSharing("your_token", your_sharing_id, true);


GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didAcceptDeviceSharing(GizWifiErrorCode result, int sharingID) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 接受成功
} else {
```

	// 接受失败
	}
	}
	};

【checkDeviceSharingInfoByQRCode】

定义	public static void checkDeviceSharingInfoByQRCode(String token, String QRCode)	
功能描述	查看二维码邀请信息。owner 用户不能查看二维码邀请信息	
参数	token	用户 token
	QRCode	二维码邀请内容。App 扫描邀请二维码时，按照以下格式解析出 type 和 code 内容：type=share&code=xxxxxxxxxx。把解析出来的 code 内容传入此参数
回调	public void didCheckDeviceSharingInfoByQRCode(GizWifiErrorCode result, String userName, String productName, String deviceAlias, String expiredAt)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	userName	创建分享邀请的 owner 用户名
	productName	设备的产品名称
	deviceAlias	设备的别名
	expiredAt	分享邀请的过期时间

代码示例	GizDeviceSharing.setListener(mListener);
	
	// 查看扫码邀请信息
	GizDeviceSharing.checkDeviceSharingInfoByQRCode("your_token",
	"your_sharing_code");
	
	GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
	@Override
	public void didCheckDeviceSharingInfoByQRCode(GizWifiErrorCode result, String
	userName, String productName, String deviceAlias, String expiredAt) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功
	} else {
	// 失败
	}
	}
	};

【acceptDeviceSharingByQRCode】

定义	public static void acceptDeviceSharingByQRCode(String token, String QRCode)
功能描述	接受二维码分享邀请。owner 用户不能调用此接口

参数	token	用户 token
	QRCode	二维码邀请内容。App 扫描邀请二维码时，按照以下格式解析出 type 和 code 内容：type=share&code=xxxxxxxxxx。把解析出来的 code 内容传入此参数
回调	public void didAcceptDeviceSharingByQRCode(GizWifiErrorCode result)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例	GizDeviceSharing.setListener(mListener);
	
	// 接受二维码分享邀请
	GizDeviceSharing.acceptDeviceSharingByQRCode("your_token",
	"your_sharing_code");
	
	GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
	@Override
	public void didAcceptDeviceSharingByQRCode(GizWifiErrorCode result) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功
	} else {
	// 失败
	}	

	}
	
	};

【modifySharingInfo】

定义	public static void modifySharingInfo(String token, int sharingID, String sharingAlias)	
功能描述	修改分享邀请别名	
参数	token	用户 token
	sharingID	要修改的分享邀请 ID
	sharingAlias	要修改的分享邀请别名
回调	public void didModifySharingInfo(GizWifiErrorCode result, int sharingID)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sharingID	修改别名的分享邀请 ID

代码示例	GizDeviceSharing.setListener(mListener);
	
	// 修改分享邀请别名
	GizDeviceSharing.modifySharingInfo("your_token", your_sharing_id, "your_sharing_alias");
	
	GizDeviceSharingListener mListener = new GizDeviceSharingListener() {	

```
@Override
public void didModifySharingInfo(GizWifiErrorCode result, int sharingID) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 成功
} else {
// 失败
}
}
};
```

【queryMessageList】

定义	public static void queryMessageList(String token, GizMessageType messageType)	
功能描述	查询消息列表。可查询分享消息	
参数	token	用户 token
	messageType	要查询的消息类型，见 GizMessageType 枚举定义
回调	public void didQueryMessageList(GizWifiErrorCode result, List<GizMessage> messageList)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	messageList	查询的消息列表
代码示例	GizDeviceSharing.setListener(mListener);	

```
// 查询消息列表
GizDeviceSharing.queryMessageList("your_token", your_sharing_id, "your_sharing_
alias", GizMessageType.GizMessageSharing);

GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
@Override
public void didModifySharingInfo(GizWifiErrorCode result, String sharingID) {
if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
// 成功
} else {
// 失败
}
}
};
```

【markMessageStatus】

定义	public static void markMessageStatus(String token, String messageID, GizMessageStatus messageStatus)	
功能描述	标记消息已读或删除	
参数	token	用户 token

	messageID	要标记或删除的消息 ID
	messageStatus	标记为已读或者删除，见 GizMessageStatus 枚举定义
回调	public void didMarkMessageStatus(GizWifiErrorCode result, String messageID)	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	messageID	标记已读或删除的消息 ID

代码示例	GizDeviceSharing.setListener(mListener);
	
	// 标记已读
	GizDeviceSharing.markMessageStatus("your_token", "your_message_id",
	GizMessageStatus.GizMessageRead);
	
	GizDeviceSharingListener mListener = new GizDeviceSharingListener() {
	@Override
	public void didModifySharingInfo(GizWifiErrorCode result, String sharingID) {
	if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功
	} else {
	// 失败
	}
	}	

	};

7. GizDeviceSharingInfo 类

7.1. 简介

GizDeviceSharingInfo 类是设备分享信息类。

7.2. 属性

属性	描述
id	int 类型，只读不可写。设备分享 ID，设备分享创建成功时会被分配一个 ID
deviceID	String 类型，只读不可写。设备 ID
productName	String 类型，只读不可写。设备的产品名称
deviceAlias	String 类型，只读不可写。设备别名
userInfo	GizUserInfo 类对象，只读不可写。这条分享邀请的账号信息，分享者或者被分享者的账号信息
alias	String 类型，只读不可写。这条分享邀请的别名
type	GizDeviceSharingType 枚举类型，只读不可写。分享邀请是分享给自己的还是自己分享给别人的
way	GizDeviceSharingWay 枚举类型，只读不可写。分享邀请是账号分享还是二维码分享
status	GizDeviceSharingStatus 枚举类型，只读不可写。分享邀请的状态，是被接受还是被拒绝的，或者还未接受
createdAt	String 类型，只读不可写。分享邀请的创建时间

属性	描述
updatedAt	String 类型，只读不可写。分享邀请的更新时间
expiredAt	String 类型，只读不可写。分享邀请的超时时间

8. GizMessage 类

8.1. 简介

GizMessage 类是机智云消息类。

8.2. 属性

属性	描述
id	String 类型，只读不可写。消息 ID
type	GizMessageType 枚举类型，只读不可写。消息类型，是系统消息还是分享消息
status	GizMessageStatus 枚举类型，只读不可写。消息状态，是否是已读、未读或已删除消息
createdAt	String 类型，只读不可写。消息生成时间
updatedAt	String 类型，只读不可写。消息更新时间
content	String 类型，只读不可写。消息内容

9. GizDeviceOTA 类

9.1. 简介

GizDeviceOTA 类提供设备固件升级功能。可升级设备的 wifi 模组固件以及 mcu 固件。

9.2. 属性访问

以下是 GizDeviceOTA 类提供的所有属性变量：

【setListener】

定义	public static void setListener(GizDeviceOTAListener listener);	
功能描述	设置 GizDeviceOTA 监听	
参数	listener	GizDeviceOTAListener 监听对象
返回值	无	
代码示例	GizDeviceSharing.setListener(mListener);	

9.3. 回调接口

以下是 GizDeviceOTA 类提供的所有回调接口：

- didNotifyDeviceUpdate：设备固件有更新的主动通知
- didNotifyDeviceUpgradeStatus：固件升级状态的主动通知
- didCheckDeviceUpdate：检查固件更新的回调
- didUpgradeDevice：固件开始升级的回调

【didNotifyDeviceUpdate】

回调	public void didNotifyDeviceUpdate(GizWifiDevice device, ConcurrentHashMap<String,String> wifiVersion, ConcurrentHashMap<String,String> mcuVersion);

回调说明	设备固件有更新的主动通知。设备固件有新版本时触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	wifiVersion	模组固件版本，字典格式：{"latest":"xxx", "current":"xxx"}。若此参数为 null，表示没有检查到模组固件更新信息
	mcuVersion	mcu 固件版本，字典格式：{"latest":"xxx", "current":"xxx"}。若此参数为 null，表示没有检查到 mcu 固件更新信息

代码示例	GizDeviceOTA.setListener(mListener);
	
	// 实现回调
	GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
	@Override
	public void didNotifyDeviceUpdate(GizWifiDevice device, ConcurrentHashMap<String,String> wifiVersion, ConcurrentHashMap<String,String> mcuVersion) {
	
	}
	};	

【didNotifyDeviceUpgradeStatus】

回调	public void didNotifyDeviceUpgradeStatus(GizWifiDevice device, GizOTAFirmwareType firmwareType, GizWifiErrorCode upgradeStatus);

回调说明	设备升级状态的主动通知。设备在升级过程中会主动上报升级状态，此时会触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	firmwareType	正在升级的固件类型
	upgradeStatus	设备升级状态，见 GizWifiErrorCode 定义中枚举值范围[8350, 8360]

代码示例	GizDeviceOTA.setListener(mListener);
	
	// 实现回调
	GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
	@Override
	public void didNotifyDeviceUpgradeStatus(GizWifiDevice device, GizOTAFirmwareType firmwareType, GizWifiErrorCode upgradeStatus) {
	
	}
	};	

【didCheckDeviceUpdate】

回调	public void didCheckDeviceUpdate(GizWifiDevice device, GizWifiErrorCode result, ConcurrentHashMap<String,String> wifiVersion, ConcurrentHashMap<String,String> mcuVersion);
回调说明	检查设备更新的回调，调用检查更新接口checkDeviceUpdate时触发该回调

回调参数	device	回调的 GizWifiDevice 对象
	result	接口执行结果，见 GizWifiErrorCode 定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时参数 wifiVersion 和 mcuVersion 值为 null
	wifiVersion	模组固件版本，字典格式：{"latest":"xxx", "current":"xxx"}。若此参数为 null，表示没有检查到模组固件更新信息
	mcuVersion	mcu 固件版本，字典格式：{"latest":"xxx", "current":"xxx"}。若此参数为 null，表示没有检查到 mcu 固件更新信息

代码示例	GizDeviceOTA.setListener(mListener);
	
	// 实现回调
	GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
	@Override
	public void didCheckDeviceUpdate(GizWifiDevice device, GizWifiErrorCode result, ConcurrentHashMap<String,String> wifiVersion, ConcurrentHashMap<String,String> mcuVersion) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功处理，取最新版本号
	} else {
	// 失败处理
	}
	}	

	};

【didUpgradeDevice】

回调	public void didUpgradeDevice(GizWifiDevice device, GizWifiErrorCode result, GizOTAFirmwareType firmwareType);	
回调说明	设备开始升级的回调，调用开始升级接口upgradeDevice时触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	result	接口执行结果，见 GizWifiErrorCode 定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	firmwareType	正在升级的固件类型

代码示例	GizDeviceOTA.setListener(mListener);
	
	// 实现回调
	GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
	@Override
	public void didUpgradeDevice(GizWifiDevice device, GizWifiErrorCode result, GizOTAFirmwareType firmwareType) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功
	} else {
	// 失败	

	}
	}
	};

9.4. API

【checkDeviceUpdate】

定义	public static void checkDeviceUpdate(String uid, String token, GizWifiDevice device);	
功能描述	检查固件是否有更新	
参数	uid	用户 uid
	token	用户 token
	device	待检查固件版本的设备

代码示例	// 设置 OTA 监听
	GizDeviceOTA.setListener(mListener);
	
	// 检查固件版本是否有更新。mDevice 为从设备列表中取到的待升级的设备
	GizDeviceOTA.checkDeviceUpdate("your_uid", "your_token", mDevice);
	
	// 实现回调
	GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
	@Override	

	public void didCheckDeviceUpdate(GizWifiDevice device, GizWifiErrorCode result, ConcurrentHashMap<String,String> wifiVersion, ConcurrentHashMap<String,String> mcuVersion) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 成功处理，取最新版本号
	} else {
	// 失败处理
	}
	}
	};

【upgradeDevice】

定义	public static void upgradeDevice(String uid, String token, GizWifiDevice device, GizOTAFirmwareType firmwareType);	
功能描述	开始升级	
参数	uid	用户 uid
	token	用户 token
	device	要升级的设备
	firmwareType	要升级的固件类型，见 GizOTAFirmwareType 枚举定义

代码示例	// 设置 OTA 委托
	GizDeviceOTA.setListener(mListener);	

```
// 开始升级。mDevice 为刚检查过版本信息待升级的设备
GizDeviceOTA.upgradeDevice("your_uid", "your_token", mDevice,
GizOTAFirmwareType.GizOTAFirmareModule);

// 实现回调
GizDeviceOTAListener mListener = new GizDeviceOTAListener() {
@Override
public void didUpgradeDevice(GizWifiDevice device, GizWifiErrorCode result,
GizOTAFirmwareType firmwareType) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 成功
    } else {
        // 失败
    }
}
};
```

10. GizDeviceGroupCenter 类

10.1. 简介

GizDeviceGroupCenter 类为 APP 开发者提供中控子设备分组操作，包括创建设备分组、删除、

更新设备分组列表等功能。

10.2. 属性访问

以下是 GizDeviceGroupCenter 类提供的所有属性变量访问：

【setListener】

定义	public static void setListener(GizDeviceGroupCenterListener listener);	
功能描述	设置 GizDeviceGroupCenter 监听	
参数	listener	GizDeviceGroupCenterListener 监听对象
返回值	无	
代码示例	GizDeviceGroupCenter.setListener(mListener);	

【getGroupListGateway】

定义	public static List<GizDeviceGroup>> getGroupListGateway(GizWifiDevice owner);	
功能描述	获取指定网关设备上的分组列表	
参数	groupOwner	中控设备对象，此参数不能填 null
返回值	组列表，GizDeviceGroup 对象数组	
代码示例	// mDevice是在设备列表中得到的网关设备对象 List<GizDeviceGroup>> list = GizDeviceGroupCenter.getGroupListGateway(mDevice);	

10.3. 回调接口

以下是 GizDeviceGroupCenter 类提供的所有回调接口：

• didUpdateSubDevices：中控子设备列表回调

【didUpdateGroups】

定义	public void didUpdateGroups(GizWifiDevice groupOwner, GizWifiErrorCode result, List<GizDeviceGroup> groupList)	
功能描述	组列表回调接口。调用添加组接口 addGroup、删除组接口 removeGroup、同步更新组列表接口 updateGroups、组列表变化上报时触发该回调	
回调参数	groupOwner	触发回调的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表；其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	groupList	组列表。GizDeviceGroup 对象数组

代码示例	GizDeviceGroupCenter.setListener(mListener);
	
	// 实现回调
	GizDeviceGroupCenterListener mListener = new GizDeviceGroupCenterListener()
	{
	@Override
	public void didUpdateGroups(GizWifiDevice groupOwner, GizWifiErrorCode result,	

```
List<GizDeviceGroup> groupList) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的组列表
    } else {
        // 失败处理
    }
}
};
```

10.4. API

【addGroup】

定义	public static void addGroup(GizWifiDevice groupOwner, String groupType, String groupName, List<GizWifiDevice> groupDevices);	
功能描述	添加分组。添加成功后会被分配一个组 ID，同时返回最新的分组列表，添加失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 null 或无效无法添加分组
	groupType	分组类型，即设备的产品唯一标识 productKey。此参数必填，填 null 或无效无法添加分组
	groupName	组名称。此参数可选填，App 可以在成功创建组以后再修改组名称
	groupDevices	组设备列表，是 GizWifiDevice 对象数组。此参数可选填， App 可以在添加组以后再添加组设备
代码示例	// mOwner为中控设备，mDevice为中控子设备列表中要加入到分组中的设备对象	

```
GizDeviceGroupCenter.setListener(mListener);

List<GizWifiDevice> list = new ArrayList<GizWifiDevice>();

list.add(mDevice);

GizDeviceGroupCenter.addGroup(mOwner, "your_product_key",

"your_group_name", list);



// 实现回调

GizDeviceGroupCenterListener mListener = new GizDeviceGroupCenterListener()

{

@Override

public void didUpdateGroups(GizWifiDevice groupOwner, GizWifiErrorCode result,

List<GizDeviceGroup> groupList) {

    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {

        // 接收变更的组列表

    } else {

        // 失败处理

    }

}

};
```

【removeGroup】

定义	public static void removeGroup(GizWifiDevice groupOwner, GizDeviceGroup group);	
功能描述	删除设备分组。删除成功时返回最新的组列表，删除失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 null 或无效无法删除分组
	group	待删除的组。此参数不能填 null

代码示例

```
// mOwner为中控设备，mGroup为从组列表中取到的组对象

GizDeviceGroupCenter.setListener(mListener);

GizDeviceGroupCenter.removeGroup(mOwner, mGroup);



// 实现回调

GizDeviceGroupCenterListener mListener = new GizDeviceGroupCenterListener()

{

@Override

public void didUpdateGroups(GizWifiDevice groupOwner, GizWifiErrorCode result,

List<GizDeviceGroup> groupList) {

    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {

        // 接收变更的组列表

    } else {

        // 失败处理

    }

}
```

	};

【updateGroups】

定义	public static void updateGroups(GizWifiDevice groupOwner);	
功能描述	更新分组列表。更新成功时返回最新的组列表，更新失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 nil 或无效无法更新分组列表

代码示例

```
// mOwner为中控设备
GizDeviceGroupCenter.setListener(mListener);
GizDeviceGroupCenter.updateGroups(mOwner);


// 实现回调
GizDeviceGroupCenterListener mListener = new GizDeviceGroupCenterListener()
{
@Override
public void didUpdateGroups(GizWifiDevice groupOwner, GizWifiErrorCode result,
List<GizDeviceGroup> groupList) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的组列表
    } else {
        // 失败处理
    }
```

	}
	
	};

11. GizDeviceGroup 类

11.1. 简介

GizDeviceGroup 类提供设备分组控制、编辑组设备、修改组信息等功能。

11.2. 属性访问

以下是 GizDeviceGroup 类提供的所有属性变量访问：

【setListener】

定义	public static void setListener(GizDeviceGroupListener listener);	
功能描述	设置 GizDeviceGroup 监听	
参数	listener	GizDeviceGroupListener 监听对象
返回值	无	

代码示例	// mGroup是从组列表中得到的组对象
	
	mGroup.setListener(mListener);	

【getGroupID】

定义	public String getGroupID();
功能描述	获取组 ID。组 ID 是 groupOwner 创建设备分组时分配的唯一标识
返回值	组 ID

代码示例	// mGroup是从组列表中得到的组对象 String groupID = mGroup.getGroupID();

【getGroupOwner】

定义	public GizWifiDevice getGroupOwner();
功能描述	管理组的设备，这个设备是用来创建、删除、维护组信息的
返回值	管理组的设备
代码示例	// mGroup是从组列表中得到的组对象 GizWifiDevice mOwner = mGroup.getGroupOwner();

【getGroupType】

定义	public String getGroupType();
功能描述	获取组类型。组类型即设备的 productKey。由于组是由同类型设备组成，所以组类型就是设备的产品类别唯一标识
返回值	组类型
代码示例	// mGroup是从组列表中得到的组对象 String type = mGroup.getGroupType();

【getGroupName】

定义	public String getGroupName();
功能描述	获取组名称
返回值	组名称

代码示例	// mGroup是从组列表中得到的组对象 String groupName = mGroup.getGroupName();

【getGroupDeviceList】

定义	public List<GizWifiDevice> getGroupDeviceList();
功能描述	获取组设备列表，组设备列表是 GizWifiDevice 对象数组。组设备列表缓存了添加到组里的设备
返回值	组设备列表
代码示例	// mGroup是从组列表中得到的组对象 List<GizWifiDevice> groupDevices = mGroup.getGroupDeviceList();

11.3. 回调接口

以下是 GizDeviceGroupCenter 类提供的所有回调接口：

- didUpdateGroupInfo：组信息更新回调
- didUpdateGroupDevices：组设备列表更新回调
- didWrite：组操作回调

【didUpdateGroupInfo】

定义	public void didUpdateGroupInfo(GizDeviceGroup group, GizWifiErrorCode result);	
功能描述	分组信息更新回调。调用修改组信息接口 editGroupInfo、组信息变化上报时触发该回调	
回调参数	group	触发回调的组对象

	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
// mGroup是从组列表中得到的组对象
mGroup.setListener(mListener);

// 实现回调
GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
@Override
public void didUpdateGroupInfo(GizDeviceGroup group, GizWifiErrorCode result) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的组信息
    } else {
        // 失败处理
    }
}
};
```

【didUpdateGroupDevices】

定义	public void didUpdateGroupDevices(GizDeviceGroup group, GizWifiErrorCode result, List<GizWifiDevice> groupDeviceList);
功能描述	组设备列表更新回调。调用添加组设备接口 addGroupDevice、删除组设备接口 removeGroupDevice、更新组设备接口 updateGroupDevices、组设备变化上报时触发该回

	调	
回调参数	group	触发回调的组对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例	// mGroup是从组列表中得到的组对象
	mGroup.setListener(mListener);
	
	// 实现回调
	GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
	@Override
	public void didUpdateGroupDevices(GizDeviceGroup group, GizWifiErrorCode result, List<GizWifiDevice> groupDeviceList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的组列表
	} else {
	// 失败处理
	}
	}
	};	

【didWrite】

定义	public void didWrite(GizDeviceGroup group, GizWifiErrorCode result, int sn);

功能描述	分组控制的回调接口。调用分组控制接口 write 时触发该回调	
回调参数	group	执行控制命令的组对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sn	控制命令的序号

代码示例	// mGroup是从组列表中得到的组对象
	mGroup.setListener(mListener);
	
	// 实现回调
	GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
	@Override
	public void didWrite(GizDeviceGroup group, GizWifiErrorCode result, int sn) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的组列表
	} else {
	// 失败处理
	}
	}
	};	

11.4. API

【editGroupInfo】

定义	public void editGroupInfo(String groupName);	
功能描述	修改设备分组信息。修改成功时返回最新的组信息，修改失败时返回错误信息	
参数	groupName	待修改的组名称。此参数不能填 null

代码示例

```
// mGroup为从组列表中取到的组对象

mGroup.setListener(mListener);



mGroup.editGroupInfo("your_group_name");



GizDeviceGroupListener mListener = new GizDeviceGroupListener() {

@Override

public void didUpdateGroupInfo(GizDeviceGroup group, GizWifiErrorCode result) {

    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {

        // 接收变更的组信息

    } else {

        // 失败处理

    }

}

};
```

【addGroupDevice】

定义	public void addGroupDevice(List<GizWifiDevice> groupDevices);	
功能描述	添加组设备。添加成功时返回最新的组设备列表，添加失败时返回错误信息	
参数	groupDevices	待添加的组设备，为 GizWifiDevice 对象数组。此参数不能填 null 或空数组

代码示例

```
// mGroup为从组列表中取到的组对象，mDevice为从中控设备子列表中取到的设备对象
mGroup.setListener(mListener);

List<GizWifiDevice> list = new ArrayList<GizWifiDevice>();
list.add(mDevice);
mGroup.addGroupDevice(list);

// 实现回调
GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
@Override
public void didUpdateGroupDevices(GizDeviceGroup group, GizWifiErrorCode
result, List<GizWifiDevice> groupDeviceList) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的组列表
    } else {
        // 失败处理
    }
}
```

	};

【removeGroupDevice】

定义	public void removeGroupDevice(List<GizWifiDevice>groupDevices);	
功能描述	删除设备分组。删除成功时返回最新的组列表，删除失败时返回错误信息	
参数	groupDevices	待删除的组设备，为 GizWifiDevice 对象数组。此参数不能填 nil 或空数组

代码示例	// mGroup为从组列表中取到的组对象，mDevice为从中控设备子列表中取到的设备对象
	mGroup.setListener(mListener);
	
	List<GizWifiDevice> list = new ArrayList<GizWifiDevice>();
	list.add(mDevice);
	mGroup.removeGroupDevice(list);
	
	// 实现回调
	GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
	@Override
	public void didUpdateGroupDevices(GizDeviceGroup group, GizWifiErrorCode result, List<GizWifiDevice> groupDeviceList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的组列表
	} else {	

```
        // 失败处理
    }
}
};
```

【updateGroupDevices】

定义	public void updateGroupDevices();
功能描述	更新分组列表。更新成功时返回最新的组列表，更新失败时返回错误信息

代码示例	// mGroup为从组列表中取到的组对象，mDevice为从中控设备子列表中取到的设备对象
	mGroup.setListener(mListener);
	mGroup.updateGroupDevices();
	
	// 实现回调
	GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
	@Override
	public void didUpdateGroupDevices(GizDeviceGroup group, GizWifiErrorCode result, List<GizWifiDevice> groupDeviceList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的组列表
	} else {
	// 失败处理

```
    }
}
};
```

【write】

定义	public void write(ConcurrentHashMap<String, Object> data, int sn);	
功能描述	执行组操作	
参数	data	待执行的组操作，数据点键值对字典，键值对为{数据点名称：数据点值}。数据点名称和值的类型要符合设备的数据点定义，透传数据的数据点名称要填"binary"，数据点值的类型要用 NSData，不符合格式的数据点参数可能无法下发。此参数不能填 nil 或空字典
	sn	控制命令序号。用于对应控制命令应答数据，控制确认回调时会返回这个 sn。如果 App 需要对应控制命令的执行顺序，sn 就要指定为一个正整数。如果 App 不关心操作执行顺序，sn 填 0。负数默认按照 0 处理

代码示例	// mGroup为从组列表中取到的组对象
	mGroup.setListener(mListener);
	
	int sn = 0;
	ConcurrentHashMap data = new ConcurrentHashMap<String, boolean> ();
	data.put("LED_OnOff", true);
	mGroup.write(data);	

```
// 实现回调
GizDeviceGroupListener mListener = new GizDeviceGroupListener() {
@Override
public void didWrite(GizDeviceGroup group, GizWifiErrorCode result, int sn) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 成功
    } else {
        // 失败
    }
}
};
```

12. GizDeviceSceneCenter 类

12.1. 简介

GizDeviceSceneCenter 类为 APP 开发者提供中控子设备分组操作，包括创建设备分组、删除、更新设备分组列表等功能。

12.2. 属性访问

以下是 GizDeviceSceneCenter 类提供的属性变量：

【setListener】

定义	public static void setListener(GizDeviceSceneCenterListener listener);
功能描述	设置 GizDeviceSceneCenter 监听

参数	listener	GizDeviceSceneCenterListener 监听对象
返回值	无	
代码示例	GizDeviceSceneCenter.setListener(mListener);	

【getSceneListGateway】

定义	public static List<GizDeviceScene>> getSceneListGateway(GizWifiDevice sceneOwner);
功能描述	获取指定中控设备上的场景列表
参数	sceneOwner：中控设备对象，此参数不能填 null
返回值	场景列表，GizDeviceScene 对象数组
代码示例	// mOwner是在设备列表中得到的中控设备对象， List<GizDeviceScene>> list = GizDeviceSceneCenter.getSceneListGateway(mDevice);

12.3. 回调接口

以下是 GizDeviceSceneCenter 类提供的所有回调接口：

- didUpdateScenes：中控场景设备列表回调

【didUpdateScenes】

定义	public void didUpdateScenes(GizWifiDevice sceneOwner, GizWifiErrorCode result, List<GizDeviceScene> sceneList);

功能描述	场景列表回调接口。调用添加场景接口 addScene、删除场景接口 removeScene、同步更新场景列表接口 updateScenes、场景列表变化上报时触发该回调	
回调参数	sceneOwner	触发回调的 GizWifiCentralControlDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表；其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	sceneList	场景列表。GizDeviceScene 对象数组

代码示例	GizDeviceSceneCenter.setListener(mListener);
	
	// 实现回调
	GizDeviceSceneCenterListener mListener = new GizDeviceSceneCenterListener() {
	@Override
	public void didUpdateScenes(GizWifiDevice sceneOwner, GizWifiErrorCode result, List<GizDeviceScene> sceneList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的组列表
	} else {
	// 失败处理
	}	

	}
	
	};

12.4. API

【addScene】

定义	public static void addScene(GizWifiDevice sceneOwner, String sceneName, List<GizDeviceSceneItem> sceneItems);	
功能描述	添加场景。添加成功后会被分配一个场景 ID，此时会回调最新的场景列表，失败时回调错误信息	
参数	sceneOwner	管理场景的设备，此参数必填，填 null 或无效无法添加分组
	sceneName	场景名称。此参数为选填参数，如果不指定场景名称此参数填 null，App 可以在成功添加场景以后再修改场景名称
	sceneItems	场景项列表，是 GizDeviceSceneItem 对象数组。此参数为选填参数，如果不指定场景项此参数填 null 或空数组，App 可以在成功添加场景以后再添加场景项

代码示例	// mOwner为中控设备，mDevice为中控子设备列表中要加入到场景项中的设备对象
	
	GizDeviceSceneCenter.setListener(mListener);
	
	ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String, Object>();
	data.put("LED_OnOff", true);
	GizDeviceSceneItem item = new GizDeviceSceneItem(mDevice, data);
	
	List<GizDeviceScene>> list = new ArrayList<GizDeviceScene>>();	

```
list.add(item);


// 添加场景
GizDeviceSceneCenter.addScene(mOwner, "your_scene_name", list];


// 实现回调
GizDeviceSceneCenterListener mListener = new GizDeviceSceneCenterListener() {
@Override
public void didUpdateScenes(GizWifiDevice sceneOwner, GizWifiErrorCode result,
List<GizDeviceScene> sceneList) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的场景列表
    } else {
        // 失败处理
    }
}
};
```

【removeScene】

定义	public static void removeScene(GizWifiDevice sceneOwner, GizDeviceScene scene);	
功能描述	删除场景。删除成功时返回最新的场景列表，删除失败时返回错误信息	
参数	sceneOwner	管理场景的设备，此参数必填，填 null 或无效无法删除场景

	scene	待删除的场景。此参数不能填 null

代码示例	// mOwner为中控设备，mScene为场景列表中待移除的场景对象
	GizDeviceSceneCenter.setListener(mListener);
	GizDeviceSceneCenter.removeScene(mOwner, mScene);
	
	// 实现回调
	GizDeviceSceneCenterListener mListener = new GizDeviceSceneCenterListener() {
	@Override
	public void didUpdateScenes(GizWifiDevice sceneOwner, GizWifiErrorCode result,
	List<GizDeviceScene> sceneList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的场景列表
	} else {
	// 失败处理
	}
	}
	};	

【updateScenes】

定义	public static void updateScenes(GizWifiDevice sceneOwner);
功能描述	更新场景列表。更新成功时返回最新的场景列表，更新失败时返回错误信息

参数	sceneOwner	管理场景的设备，此参数必填，填 nil 或无效无法更新场景列表

代码示例

```
// mOwner为中控设备
GizDeviceSceneCenter.setListener(mListener);
GizDeviceSceneCenter.updateScenes(mOwner, mScene);

// 实现回调
GizDeviceSceneCenterListener mListener = new GizDeviceSceneCenterListener() {
@Override
public void didUpdateScenes(GizWifiDevice sceneOwner, GizWifiErrorCode result,
List<GizDeviceScene> sceneList) {
    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
        // 接收变更的场景列表
    } else {
        // 失败处理
    }
}
};
```

13. GizDeviceScene 类

13.1. 简介

GizDeviceScene 类提供场景执行、场景编辑等功能。

13.2. 属性访问

以下是 GizDeviceScene 类提供的所有属性变量访问：

【setListener】

定义	public static void setListener(GizDeviceSceneListener listener);	
功能描述	设置 GizDeviceScene 监听	
参数	listener	GizDeviceSceneListener 监听对象
返回值	无	

代码示例	// mScene是从组列表中得到的场景对象
	mScene.setListener(mListener);	

【getSceneID】

定义	public String getSceneID();
功能描述	获取场景 ID。场景 ID 是 sceneOwner 添加场景时分配的唯一标识
返回值	场景 ID
代码示例	// mScene是从场景列表中得到的场景对象 String sceneID= mScene.getSceneID();

【getSceneOwner】

定义	public GizWifiDevice getSceneOwner();
功能描述	管理场景的设备，这个设备是用来创建、删除、维护场景信息的
返回值	管理场景的设备

代码示例	// mScene是从组列表中得到的组对象 GizWifiDevice mOwner = mScene.getSceneOwner();

【getSceneName】

定义	public String getSceneName();
功能描述	获取场景名称
返回值	场景名称
代码示例	// mScene是从组列表中得到的场景对象 String sceneName = mScene.getSceneName();

【getSceneItemList】

定义	public List<GiDeviceSceneItem> getSceneItemList();
功能描述	获取场景项列表，场景项列表是 GiDeviceSceneItem 对象数组。场景项列表缓存了添加到场景里的场景项
返回值	场景项列表
代码示例	// mScene是从场景列表中得到的场景对象 List<GiDeviceSceneItem> groupDevices = mScene.getSceneItemList ();

13.3. 回调接口

以下是 GizDeviceScene 类提供的所有回调接口：

- didUpdateSceneInfo：场景信息更新回调
- didUpdateSceneItems：场景项列表更新回调

• didUpdateSceneStatus：场景状态更新回调

【didUpdateSceneInfo】

定义	public void didUpdateSceneInfo(GizDeviceScene scene, GizWifiErrorCode result);	
功能描述	场景信息更新回调。调用编辑场景信息接口 editSceneInfo、场景信息变化上报时使用该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例	// mScene为从场景列表中取到的场景对象
	mScene.setListener(mListener);
	
	// 实现回调
	GizDeviceSceneListener mListener = new GizDeviceSceneListener() {
	@Override
	public void didUpdateSceneInfo(GizDeviceScene scene, GizWifiErrorCode result) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的场景信息
	} else {
	// 失败处理
	}
	}	

	};

【didUpdateSceneItems】

定义	public void didUpdateSceneItems(GizDeviceScene scene, GizWifiErrorCode result, List<GizDeviceSceneItem> sceneItemList);	
功能描述	场景项列表更新回调。调用编辑场景项接口 editSceneItems、场景项列表变化上报时触发该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功同时 sceneItemList 为最新的场景项列表，其他为失败
	sceneItemList	场景项列表，为 GizDeviceSceneItem 对象数组

代码示例	// mScene为从场景列表中取到的场景对象
	mScene.setListener(mListener);
	
	// 实现回调
	GizDeviceSceneListener mListener = new GizDeviceSceneListener() {
	@Override
	public void didUpdateSceneItems(GizDeviceScene scene, GizWifiErrorCode result, List<GizDeviceSceneItem> sceneItemList) {
	if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {
	// 接收变更的场景项列表	

```
    } else {
        // 失败处理
    }
}
};
```

【didUpdateSceneStatus】

定义	public void didUpdateSceneStatus(GizDeviceScene scene, GizWifiErrorCode result, GizDeviceSceneStatus status);	
功能描述	场景状态更新回调。调用接口 updateSceneStatus、场景执行状态更新上报时触发该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	status	场景的执行状态

代码示例	// mScene为从场景列表中取到的场景对象
	mScene.setListener(mListener);
	
	// 实现回调
	GizDeviceSceneListener mListener = new GizDeviceSceneListener() {
	@Override
	public void didUpdateSceneStatus(GizDeviceScene scene, GizWifiErrorCode result,	

```
GizDeviceSceneStatus status) {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的场景状态
    } else {
        // 失败处理
    }
}
};
```

【didExecuteScene】

定义	public void didExecuteScene(GizDeviceScene scene, GizWifiErrorCode result, int sn);	
功能描述	场景执行回调。调用场景执行接口 executeScene 时触发该回调	
回调参数	scene	场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sn	执行序号

代码示例	// mScene为从场景列表中取到的场景对象
	mScene.setListener(mListener);
	
	// 实现回调
	GizDeviceSceneListener mListener = new GizDeviceSceneListener() {	

```
@Override
public void didExecuteScene(GizDeviceScene scene, GizWifiErrorCode result, int sn)
{
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收执行结果
    } else {
        // 失败处理
    }
}
};
```

13.4. API 【editSceneInfo】

定义	public static void editSceneInfo(String sceneName);	
功能描述	编辑场景信息。编辑成功时返回最新的场景信息，编辑失败时返回错误信息	
参数	sceneName	待修改的场景名称。此参数不能填 null

代码示例

```
// mScene为从场景列表中取到的场景对象
mScene.setListener(mListener);
mScene.editSceneInfo("your_scene_name");

// 实现回调
```

```
GizDeviceSceneListener mListener = new GizDeviceSceneListener() {

@Override

public void didUpdateSceneInfo(GizDeviceScene scene, GizWifiErrorCode result) {

    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {

        // 接收变更的场景信息

    } else {

        // 失败处理

    }

}

};
```

【editSceneItems】

定义	public void editSceneItems(List<GizDeviceSceneItem> sceneItems);	
功能描述	编辑场景项列表。编辑成功时返回最新的场景项列表，失败时返回错误信息	
参数	sceneItems	待编辑的场景项列表，包含全部场景项列表，GizDeviceSceneItem 对象数组。这个列表必须能符合预期的修改结果，列表中应包括新添加的、待修改的、不修改的，如果有待删除的要移除掉。此参数不能填 nil 或空数组

代码示例

```
// mScene为从场景列表中取到的场景对象，mDevice为中控设备子列表中取到的设备对象

mScene.setListener(mListener);

ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String,
Object>();
data.put("LED_OnOff", true);
GizDeviceSceneItem item = GizDeviceSceneItem(mDevice, data);

List<GizDeviceScene>> list = new ArrayList<GizDeviceScene>>();
```

```
list.add(item);

// 添加场景

mScene.editSceneItems(list);



// 实现回调

GizDeviceSceneListener mListener = new GizDeviceSceneListener() {

@Override

public void didUpdateSceneItems(GizDeviceScene scene, GizWifiErrorCode result,

List<GizDeviceSceneItem> sceneItemList) {

    if(result == GizWifiErrorCode.GIZ_SDK_SUCCESS) {

        // 接收变更的场景项列表

    } else {

        // 失败处理

    }

}

};
```

【executeScene】

定义	public void executeScene(boolean startup, int sn)	
功能描述	启动或取消场景执行	
参数	startup	执行或取消执行。YES 为执行，NO 为取消执行

	sn	执行序号。用于对应执行应答数据，执行确认回调时会返回这个 sn。 如果 App 需要对应执行的执行顺序，sn 就要指定为一个正整数。如果 App 不关心操作执行顺序，sn 填 0。负数默认按照 0 处理
代码示例		

```
// mScene为从场景列表中取到的场景对象

mScene.setListener(mListener);

ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String,
Object>();
data.put("LED_OnOff", true);

mScene.executeScene(data);



// 实现回调

GizDeviceSceneListener mListener = new GizDeviceSceneListener() {

@Override

public void didExecuteScene(GizDeviceScene scene, GizWifiErrorCode result, int sn)

{

    if(result.code == GIZ_SDK_SUCCESS) {

        // 接收执行结果

    } else {

        // 失败处理

    }

}

};
```

【updateSceneStatus】

定义	public void updateSceneStatus();
功能描述	更新场景的执行状态

代码示例	// mScene为从场景列表中取到的场景对象
	mScene.setListener(mListener);
	mScene.updateSceneStatus();
	
	// 实现回调
	GizDeviceSceneListener mListener = new GizDeviceSceneListener() {
	@Override
	public void didUpdateSceneStatus(GizDeviceScene scene, GizWifiErrorCode result,
	GizDeviceSceneStatus status) {
	if(result.code == GIZ_SDK_SUCCESS) {
	// 接收变更的场景状态
	} else {
	// 失败处理
	}
	}
	};

14. GizDeviceSceneItem 类

14.1. 简介

GizDeviceSceneItem 提供场景项信息。场景项分为设备场景项、组场景项和延时场景项三种类型，设备场景项和组场景项必须要设置数据点操作，延时场景项只需要设置延时时间不需要设置数据点操作

14.2. 属性访问

以下是 GizDeviceSceneItem 类提供的所有属性变量访问：

【getDevice】

定义	public GizWifiDevice getDevice();
功能描述	设备场景项的中控子设备，如果 sceneItemType 是设备场景项，需要关心此变量
返回值	设备场景项的设备对象
代码示例	// mSceneItem是从场景项列表中得到的场景项对象 GizWifiDevice device = mSceneItem.getDevice();

【getGroup】

定义	public GizDeviceGroup getGroup();
功能描述	组场景项的中控组，如果 sceneItemType 是组场景项，需要关心此变量
返回值	组场景项的中控组对象
代码示例	// mSceneItem是从场景项列表中得到的场景项对象 GizWifiGroup group = mSceneItem.getGroup();

【getData】

定义	public ConcurrentHashMap<String, Object> getData();
功能描述	获取场景项操作
返回值	场景项操作
代码示例	// mSceneItem是从场景项列表中得到的场景项对象 ConcurrentHashMap<String, Object> data = mSceneItem.getData();

【getDelayTime】

定义	public int getDelayTime();
功能描述	延时的毫秒数，如果 sceneItemType 是延时场景项，需要关心此变量。延时场景项不关注其他变量
返回值	延时时间
代码示例	// mSceneItem是从场景项列表中得到的场景项对象 int delayTime = mSceneItem.getDelayTime();

【getSceneItemType】

定义	public GizSceneItemType getSceneItemType();
功能描述	场景项类型，见 GizSceneItemType 枚举定义
返回值	场景项类型
代码示例	// mSceneItem是从场景项列表中得到的场景项对象 GizSceneItemType type = mSceneItem.getSceneItemType();

14.3. API

【GizDeviceSceneItem】(设备场景项)

定义	public GizDeviceSceneItem(GizWifiDevice device, ConcurrentHashMap<String, Object> data);
功能描述	构造函数，用于创建设备场景项对象

代码示例	// mDevice为中控子设备列表中要加入到场景项中的设备对象
	ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String, Object>();
	data.put("LED_OnOff", true);
	GizDeviceSceneItem item = new GizDeviceSceneItem(mDevice, data);

【GizDeviceSceneItem】(组场景项)

定义	public GizDeviceSceneItem(GizDeviceGroup group, ConcurrentHashMap<String, Object> data);
功能描述	构造函数，用于创建组场景项对象

代码示例	// mGroup为中控组列表中要加入到场景项中的组对象
	ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String, Object>();
	data.put("LED_OnOff", true);
	GizDeviceSceneItem item = new GizDeviceSceneItem(mGroup, data);

【GizDeviceSceneItem】(延时场景项)

定义	public GizDeviceSceneItem(int delayTime);
功能描述	构造函数，用于创建延时场景项对象
代码示例	GizDeviceSceneItem item = new GizDeviceSceneItem(10);

15. GizDeviceSchedulerCenter 类

15.1. 简介

GizDeviceSchedulerCenter 类为 APP 开发者提供设备定时任务管理功能，管理用户在设备上设置的定时任务。

15.2. 属性访问

以下是 GizDeviceSchedulerCenter 类提供的属性变量访问：

【setListener】

定义	public void setListener(GizDeviceSchedulerCenterListener listener)	
功能描述	设置定时任务监听	
参数	listener	定时任务监听
代码示例	GizDeviceSchedulerCenter.setListener(mListener);	

【getSchedulerListCloud】

定义	public static List<GizDeviceScheduler> getSchedulerListCloud(GizWifiDevice owner);	
功能描述	获取云端定时任务列表，这个列表缓存了 GizDeviceScheduler 对象	
参数	schedulerOwner	执行定时任务的设备，此参数不能填 null
返回值	定时任务列表。参数值为 null 或无效设备时返回空数组	
代码示例	// mOwner是在设备列表中得到的设备对象， List<GizDeviceScheduler> list =	

	GizDeviceSchedulerCenter.getSchedulerListCloud(mOwner);

15.3. 回调接口

以下是 GizDeviceSchedulerCenter 类提供的所有回调接口：

- didUpdateSchedulers：定时任务列表回调

【didUpdateSchedulers】

定义	public void didUpdateSchedulers(GizWifiDevice schedulerOwner, GizWifiErrorCode result, List<GizDeviceScheduler> schedulerList)	
功能描述	定时任务列表回调接口。调用创建定时任务接口 createScheduler、删除定时任务接口 deleteScheduler、同步更新定时任务列表接口 updateSchedulers、定时任务列表变化上报时该回调	
回调参数	schedulerOwner	触发回调的 GizWifiDevice 设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 schedulerList 为定时任务列表；其他为失败，此时 schedulerList 大小为 0
	schedulerList	设备定时任务列表，GizDeviceSchedulerSuper 对象数组，GizDeviceSchedulerSuper 是 GizDeviceScheduler 的父类。处理云端定时任务时，App 可以用 GizDeviceScheduler 类对象处理定时任务列表
代码示例	GizDeviceSchedulerCenter.setListener(mListener);	

```
// 实现回调
GizDeviceSchedulerCenterListener mListener = new GizDeviceSchedulerCenterListener() {
@Override
public void didUpdateSchedulers(GizWifiErrorCode result, GizWifiDevice schedulerOwner, List<GizDeviceScheduler> schedulerList) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
        // 接收变更的定时任务列表
    } else {
        // 失败处理
    }
}
};
```

15.4. API 【createScheduler】

定义	public static void createScheduler(String uid, String token, GizWifiDevice schedulerOwner, GizDeviceSchedulerSuper scheduler, List<GizDeviceSchedulerTask> schedulerTasks);
功能描述	创建定时任务。创建成功会被分配一个定时任务 ID，同时通过回调接口 didUpdateSchedulers

	给 App 返回定时任务列表，创建失败时返回错误信息	
参数	uid	用户 uid。创建云端定时任务时此参数必填，空串和无效值无法创建云端定时任务
	token	用户 token。创建云端定时任务时此参数必填，空串和无效值无法创建云端定时任务
	schedulerOwner	管理定时任务的设备，此参数必填，填 null 无法创建定时任务
	scheduler	定时任务对象，用于填写定时任务内容的，此参数不能填 null。 云端定时任务对象通过 GizDeviceScheduler 类的构造函数创建， GizDeviceScheduler 类提供三种构造方法，App 根据需要选择对应的方法创建对象
	schedulerTasks	创建云端定时任务时不需要填写此参数

代码示例	// 设置定时任务监听
	GizDeviceSchedulerCenter.setListener(mListener);
	
	// 一次性定时任务，在 2017 年 1 月 16 日早上 6 点 30 分开灯
	ConcurrentHashMap<String, Object> data = new ConcurrentHashMap<String, Object>();
	data.put("LED_OnOff", true);
	GizDeviceScheduler scheduler = new GizDeviceScheduler(data, "2017-01-16", "06:30", true, "开灯任务");
	
	// 创建云端定时任务，mDevice 为从设备列表中取到的要创建定时任务的设备对象	

```
GizDeviceSchedulerCenter.createScheduler("your_uid", "your_token", mDevice, scheduler, null);


// 实现回调
GizDeviceSchedulerCenterListener mListener = new GizDeviceSchedulerCenterListener() {
@Override
public void didUpdateSchedulers(GizWifiErrorCode result, GizWifiDevice schedulerOwner, List<GizDeviceScheduler> schedulerList) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
        // 接收变更的定时任务列表
    } else {
        // 失败处理
    }
}
};
```

【deleteScheduler】

定义	public static void deleteScheduler(String uid, String token, GizWifiDevice schedulerOwner, GizDeviceSchedulerSuper scheduler);
功能描述	删除定时任务。删除成功时通过回调接口 didUpdateSchedulers 给 App 返回定时任务列表，删除失败时返回错误信息

参数	uid	用户 uid。删除云端定时任务时此参数必填，空串和无效值无法删除云端定时任务
	token	用户 token。删除云端定时任务时此参数必填，空串和无效值无法删除云端定时任务
	schedulerOwner	管理定时任务的设备，此参数必填，填 null 无法删除定时任务
	scheduler	待删除的定时任务对象，是从定时任务列表中得到的 GizDeviceScheduler 对象，此参数必填，不能填 null

代码示例

```
// 设置定时任务监听
GizDeviceSchedulerCenter.setListener(mListener);


// 删除设备的定时任务，mDevice为从设备列表中取到的要删除定时任务的设备对象
// mScheduler为从定时任务列表中得到的待删除的定时任务对象
GizDeviceSchedulerCenter.deleteScheduler("your_uid", "your_token", mDevice, mScheduler);


// 实现回调
GizDeviceSchedulerCenterListener mListener = new GizDeviceSchedulerCenterListener() {
@Override
public void didUpdateSchedulers(GizWifiErrorCode result, GizWifiDevice schedulerOwner, List<GizDeviceScheduler> schedulerList) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
```

```
        // 接收变更的定时任务列表
    } else {
        // 失败处理
    }
}
};
```

【updateSchedulers】

定义	public static void updateSchedulers(String uid, String token, GizWifiDevice schedulerOwner);	
功能描述	同步更新设备定时任务列表。成功时通过回调接口 didUpdateSchedulers 返回同步更新结果，失败时返回错误信息	
参数	uid	用户 uid。同步更新云端定时任务时此参数必填
	token	用户 token。同步更新云端定时任务时此参数必填
	schedulerOwner	管理定时任务的设备，此参数必填

代码示例	// 设置定时任务监听
	GizDeviceSchedulerCenter.setListener(mListener);
	
	// 与设备同步定时任务，mDevice 为从设备列表中取到的要同步更新定时任务的设备对象
	GizDeviceSchedulerCenter.updateSchedulers("your_uid", "your_token", mDevice);	

```
// 实现回调
GizDeviceSchedulerCenterListener mListener = new GizDeviceSchedulerCenterListener() {
@Override
public void didUpdateSchedulers(GizWifiErrorCode result, GizWifiDevice schedulerOwner, List<GizDeviceScheduler> schedulerList) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
        // 接收变更的定时任务列表
    } else {
        // 失败处理
    }
}
};
```

16. GizDeviceScheduler 类

16.1. 简介

GizDeviceScheduler 类是云端定时任务类，可设置一次性定时任务、按月重复的定时任务、按周重复的定时任务。一次性定时任务是指只执行一次定时任务，按月重复定时任务是指在每月特定日期执行定时任务，按周重复定时任务是指在每周特定时间执行定时任务。

该类继承自父类 GizDeviceSchedulerSuper，GizDeviceSchedulerCenter 类回调定时任务列表时用 GizDeviceSchedulerSuper 对象做回调。App 处理定时任务列表时需要判断数组内的对象类型是否为 GizDeviceScheduler 类。

16.2. 属性访问

属性	描述
schedulerID	NSString 类型，只读不可写。定时任务 ID，定时任务创建成功时会被分配一个 ID
schedulerOwner	GizWifiDevice 类型，只读不可写。管理定时任务的设备
date	NSString 类型，可读写。定时任务的执行日期，年月日以“－”符号分割，例如：2017-01-30。此变量默认值为 nil
time	NSString 类型，可读写。定时任务的执行时间，24 小时制，例如：06:30。无论是一次性定时任务还是按周、按月重复的定时任务，必须指定执行时间，此变量必须有值
weekdays	GizScheduleWeekday 枚举类型数组，可读写。此变量为 nil，表示定时任务不需要按周重复。需要按周重复时，此变量可填写为一周中的某一天或者某几天，例如：想在周一和周三执行定时任务，就把 GizScheduleMonday、GizScheduleWednesday 这两个值放到数组里。此变量默认值为 nil。 说明：定时任务是按周重复还是按月重复，只能二选一。如果定时任务是按周重复，则忽略按月重复。例如：weekdays 和 monthDays 都被赋值时，会优先取 weekdays 的值
monthDays	NSInteger 类型数组，可读写。此变量为 nil，表示定时任务不需要按月重复。需要按月重复时，此变量可填写为一个月中的某一天或某几天。例如，想在一个月的 1 号和 15 号执行定时任务，可把 1、15 这两个值放到数组里。此变量默认值为 nil。 说明：定时任务是按周重复还是按月重复，只能二选一。如果定时任务是按月重复，需要把按周重复变量 weekdays 清空，按月重复才能生效
enabled	BOOL 类型，可读写。值为 YES 表示启动定时任务，默认值为 NO
remark	NSString 类型，可读写。定时任务备注信息，默认值为 nil

属性	描述
startDate	NSString 类型，可读写。定时任务启动日期，年月日以“－”符号分割。格式为：2017-01-29。默认值为 nil
endDate	NSString 类型，可读写。定时任务结束日期，年月日以“－”符号分割。格式为：2017-02-01。默认值为 nil
createdDateTime	NSString 类型，只读不可写。定时任务的创建时间
attrs	NSDictionary 类型，可读写。定时任务要执行的动作，为数据点名称和值的键值对。定时任务必须指定要执行的动作，否则无法创建定时任务。此变量必须有值
schedulerType	云端定时任务有三种类型：一次性任务、按周重复、按天重复，是根据时间重复类型区分的。一次性任务需要同时设置 date、time，按周重复需要设置 time 和 weekdays，按天需要设置 time 和 monthDays；

以下是 GizDeviceScheduler 类提供的属性变量访问：

【setListener】

定义	public void setListener(GizDeviceSchedulerListener listener)	
功能描述	设置定时任务监听	
参数	listener	定时任务监听

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setListener(mListener);	

【getSchedulerID】

定义	public String getSchedulerID();

功能描述	定时任务 ID，是定时任务创建时被分配的唯一标识
返回值	定时任务 ID
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String scnedulerID = mScheduler.getSchedulerID();

【getSchedulerOwner】

定义	public GizWifiDevice getSchedulerOwner();
功能描述	管理定时任务的设备，是用于创建、删除、维护定时任务信息的
返回值	设备对象
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 GizWifiDevice mOwner = mScheduler.getSchedulerOwner();

【getDate】

定义	public String getDate();
功能描述	定时任务的预设日期，格式形如：1990-10-03。定时任务会在预设日期这一天到达时执行
返回值	预设日期
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String date = mScheduler.getDate();

【setDate】

定义	public void setDate(String date);

功能描述	定时任务的预设日期。定时任务会在预设日期这一天到达时执行	
参数	date	预设日期，格式形如：1990-10-03
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setDate("1990-10-03");	

【getTime】

定义	public String getTime();
功能描述	定时任务的预设时间，24 小时制，格式形如：07:08。定时任务会在预设时间到达时执行
返回值	预设时间
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String time = mScheduler.getTime();

【setTime】

定义	public void setTime(String time);	
功能描述	定时任务的预设时间。定时任务会在预设时间到达时执行	
参数	time	预设时间，24 小时制，格式形如：07:08
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setTime("07:08");	

【getWeekdays】

定义	public List<GizScheduleWeekday> getWeekdays();
功能描述	按周重复，定时任务会每周重复执行
返回值	按周重复的日期
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 List<GizScheduleWeekday> weekdays = mScheduler.getWeekdays();

【setWeekdays】

定义	public void setWeekdays(List<GizScheduleWeekday> weekdays);	
功能描述	按周重复，定时任务会每周重复执行	
参数	weekdays	预设重复日期
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	List<GizScheduleWeekday> weekdays = new ArrayList<GizScheduleWeekday>();
	weekdays.add(GizScheduleWeekday. GizScheduleMonday);
	mScheduler.setWeekdays(weekdays);	

【getMonthDays】

定义	public List<Integer> getMonthDays();
功能描述	按天重复，定时任务会每周重复执行
返回值	按天重复的日期
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象

	List<Integer> monthDays = mScheduler.getMonthDays();

【setMonthDays】

定义	public void setMonthDays(List<Integer> monthDays);	
功能描述	按周重复，定时任务会每周重复执行	
参数	weekdays	预设重复日期
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	List<Integer> monthDays= new ArrayList<Integer>();
	weekdays.add(1);
	weekdays.add(15);
	mScheduler.setWeekdays(monthDays);	

【getEnabled】

定义	public boolean getEnabled();
功能描述	定时任务是否开启。true 表示开启，false 表示关闭
返回值	是否开启
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 boolean enabled = mScheduler.getEnabled();

【setEnabled】

定义	public void setEnabled(boolean enabled);	
功能描述	定时任务是否开启	
参数	enabled	true 表示开启，false 表示关闭
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setEnabled(true);	

【getSchedulerType】

定义	GizSchedulerType getSchedulerType();
功能描述	定时任务类型。见 GizSchedulerType。
返回值	定时任务类型
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 GizSchedulerType type = mScheduler.getSchedulerType();

【getRemark】

定义	String getRemark();
功能描述	定时任务的备注信息
返回值	定时任务的备注信息
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String remark = mScheduler.getRemark();

【setRemark】

定义	public void setRemark(String remark);	
功能描述	定时任务的备注信息	
参数	remark	备注信息
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setRemark("your_remark");	

【getAttrs】

定义	public <ConcurrentHashMap<String, Object> getAttrs();
功能描述	定时任务要执行的动作，数据点键值对字典，键值对为{数据点名称：数据点值}。数据点名称和值的类型要符合设备的数据点定义，扩展数据点值的类型要用 NSData。
返回值	定时任务预设的结束时间
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 <ConcurrentHashMap<String, Object> data = mScheduler. getAttrs();

【setAttrs】

定义	public void setAttrs(ConcurrentHashMap<String, Object> attrs);	
功能描述	定时任务要执行的动作。定时任务必须指定要执行的动作，否则无法创建定时任务	
参数	attrs	，数据点键值对字典，键值对为{数据点名称：数据点值}。数据点名称和值的类型要符合设备的数据点定义，扩展数据点值的类型要用 NSData
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	ConcurrentHashMap data = new ConcurrentHashMap<String, boolean>();
	data.put("LED_OnOff", true);
	mScheduler.setAttrs(data);

【getStartDate】

定义	String getStartDate();	
功能描述	定时任务预设的起始时间，格式与 date 相同	
返回值	定时任务预设的起始时间	
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String startDate = mScheduler.getStartDate();	

【setStartDate】

定义	public void setStartDate(String startDate);	
功能描述	定时任务的开始时间	
参数	startDate	开始时间
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setStartDate("2017-11-03");	

【getEndDate】

定义	String getEndDate();
功能描述	定时任务预设的结束时间，格式与 date 相同
返回值	定时任务预设的结束时间
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String endDate = mScheduler.getEndDate();

【setEndDate】

定义	public void setEndDate(String endDate);	
功能描述	定时任务的结束时间	
参数	endDate	结束时间
返回值	无	

代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象
	mScheduler.setEndDate("2017-11-04");	

【getCreatedDateTime】

定义	String getCreatedDateTime();
功能描述	定时任务预设的结束时间，格式与 date 相同
返回值	定时任务预设的结束时间
代码示例	// mScheduler 是从云端定时任务列表中得到的定时任务对象 String createDateTime = mScheduler. getCreatedDateTime();

16.3. 回调接口

以下是 GizDeviceSchedulerCenter 类提供的所有回调接口，将在后续 API 定义中详细介绍：

- didUpdateSchedulerInfo：定时任务列表回调

【didUpdateSchedulerInfo】

定义	public void didUpdateSchedulerInfo(GizDeviceScheduler scheduler, GizWifiErrorCode result);	
功能描述	定时任务信息更新回调。调用修改定时任务信息接口 editSchedulerInfo、定时任务信息变化上报时触发该回调	
回调参数	scheduler	触发回调的云端定时任务对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 schedulerList 为定时任务列表；其他为失败，此时 schedulerList 大小为 0

代码示例	// mScheduler为从定时任务列表中取到的定时任务对象
	mScheduler.setListener(mListener);
	
	// 实现回调
	GizDeviceSchedulerListener mListener = new GizDeviceSchedulerListener() {
	@Override
	public void didUpdateSchedulerInfo(GizDeviceScheduler scheduler,	

```
GizWifiErrorCode result) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
        // 接收变更的定时任务信息
    } else {
        // 失败处理
    }
}
};
```

16.4. API

【GizDeviceScheduler】（一次性）

定义	public GizDeviceScheduler(ConcurrentHashMap<String, Object> attrs, String date, String time, boolean enabled, String remark);	
功能描述	构造函数，用于生成一次性云端定时任务对象	
参数	attrs	定时任务操作键值对字典：{操作名字：操作值}，请注意不支持透传数据。此参数不能填 null 或空字典
	date	定时任务的预设日期，格式形如：1990-10-03。定时任务将在预设日期这一天到达时执行。此参数不能填 null 或空串，如果填写了过去日期或者不符合约定格式，无法在云端创建定时任务
	time	定时任务的预设时间，24 小时制，格式形如：07:08。定时任务将在预设时间到达时执行。此参数不能填 null 或空串，必须符合约定格式，否则无法在云端创建定时任务

	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 null

代码示例	// 一次性定时任务，在 2017 年 1 月 16 日早上 6 点 30 分开灯
	ConcurrentHashMap data = new ConcurrentHashMap<String, boolean>();
	data.put("LED_OnOff", true);
	
	// 生成一次性定时任务对象
	GizDeviceScheduler *mScheduler = new GizDeviceScheduler(data, "2017-01-16", "06:30", true, "开灯任务");	

【GizDeviceScheduler】（按周重复）

定义	public GizDeviceScheduler(ConcurrentHashMap<String, Object> attrs, String time, List<GizScheduleWeekday> weekDays, boolean enabled, String remark);	
功能描述	构造函数，用于生成按周重复的云端定时任务对象	
参数	attrs	定时任务操作键值对字典：{操作名字: 操作值}，请注意不支持透传数据。此参数不能填 null 或空字典
	time	定时任务的预设时间，24 小时制，格式形如：07:08。此参数不能填 null 或空串，必须符合约定格式。定时任务将在预设时间到达时执行
	weekDays	按周重复，GizScheduleWeekday 数组。定时任务可以预设为每周的某几天重复执行。此参数不能填 null 或空数组，数组中重复的值会被合并，无效值会被滤除，如果滤除后数组大小为 0 按空数组处理

	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 null

代码示例	// 按周重复定时任务，每周一和周五早上 6 点 30 分开灯
	ConcurrentHashMap data = new ConcurrentHashMap<String, boolean>();
	data.put("LED_OnOff", true);
	
	List<GizScheduleWeekday> weekdays = new ArrayList<GizScheduleWeekday>();
	weekdays.add(GizScheduleWeekday.GizScheduleMonday);
	weekdays.add(GizScheduleWeekday.GizScheduleFriday);
	
	// 生成按周重复定时任务对象
	GizDeviceScheduler *mScheduler = new GizDeviceScheduler(data, "06:30", weekdays, true, "开灯任务");	

【GizDeviceScheduler】（按天重复）

定义	public GizDeviceScheduler(ConcurrentHashMap<String, Object> attrs, String time, List<GizScheduleWeekday> monthDays, boolean enabled, String remark);	
功能描述	构造函数，用于生成按天重复的云端定时任务对象	
参数	attrs	定时任务操作键值对字典：{操作名字: 操作值}，请注意不支持透传数据。此参数不能填 null 或空字典
	time	定时任务的预设时间，24 小时制，格式形如：07:08。此参数不能填

		null 或空串，必须符合约定格式。定时任务将在预设时间到达时执行
	monthDays	按天重复，整数数组。定时任务可以预设为每周的某几天重复执行。此参数不能填 null 或空数组，数组中重复的值会被合并，无效值会被滤除，如果滤除后数组大小为 0 按空数组处理
	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 null

代码示例	// 按天重复定时任务，每月 1 号和 15 号早上 6 点 30 分开灯
	ConcurrentHashMap data = new ConcurrentHashMap<String, boolean>();
	data.put("LED_OnOff", true);
	
	List<Integer> monthDays = new ArrayList<Integer>();
	monthDays.add(1);
	monthDays.add(15);
	
	// 生成按天重复定时任务对象
	GizDeviceScheduler *mScheduler = new GizDeviceScheduler(data, "06:30", monthDays, true, "开灯任务");	

【editSchedulerInfo】

定义	public void editSchedulerInfo(GizSchedulerType type);
功能描述	修改定时任务信息，此接口可用于修改云端或中控的定时任务信息。请注意，必须要先修改对应的变量值，然后再调用此接口完成修改。修改成功时返回最新的定时任务信息，修改失败时返回错误信息

参数	uid	用户 uid。此参数必填，不能填空串或无效值
	token	用户 token。此参数必填，不能填空串或无效值
	schedulerType	定时任务类型，GizSchedulerType 枚举。此参数不能填无效值

代码示例	// mScheduler 是定时任务列表中要修改的定时任务对象
	// 把之前创建好的一次性定时任务修改成每月 1 号和 15 号重复执行的定时任务，
	mScheduler.setDate("2017-01-16");
	mScheduler.setTime("06:30");
	mScheduler.setRemark("开灯任务");
	
	ConcurrentHashMap<String, Object> attrs = new ConcurrentHashMap<String, Object>();
	attrs.put("LED_OnOff", true);
	mScheduler.setAttrs(attrs);
	
	List<Integer> monthDays = new ArrayList<Integer>();
	monthDays.add(1);
	monthDays.add(15);
	mScheduler.setMonthDays(monthDays);
	
	// 设置定时任务监听
	mScheduler.setListener(mListener);	

```
// 修改设备的定时任务
mScheduler.editScheduler("your_uid", "your_token",
GizSchedulerType.GizSchedulerDayRepeat);

// 实现回调
GizDeviceSchedulerCenterListener mListener = new
GizDeviceSchedulerCenterListener() {
@Override
public void didUpdateSchedulerInfo(GizDeviceScheduler scheduler,
GizWifiErrorCode result) {
    if (result == GizWifiErrorCode.GIZ_SDK_SUCCESS)) {
        // 接收变更的定时任务信息
    } else {
        // 失败处理
    }
}
};
```

17. 枚举类定义

17.1. 简介

本节说明 GizWifiSDK 中使用的所有枚举定义。

17.2. 定义

【GizLogPrintLevel】

功能描述：日志打印级别。

枚举 ID	枚举定义	描述
0	GizLogPrintNone	不输出任何日志
1	GizLogPrintI	输出错误日志
2	GizLogPrintII	输出调试日志
3	GizLogPrintAll	输出数据日志

【GizEventType】

功能描述：事件通知类型。

枚举 ID	枚举定义	描述
0	GizEventSDK	SDK 系统事件
1	GizEventDevice	设备异常事件
2	GizEventM2MService	M2M 异常事件
5	GizEventToken	Token 失效事件

【GizWifiConfigureMode】

功能描述：设备配置模式。

枚举 ID	枚举定义	描述
0	GizWifiSoftAP	SoftAP 配置模式
1	GizWifiAirLink	AirLink 配置模式

【GizWifiDeviceType】

功能描述：设备类型。

枚举 ID	枚举定义	描述
0	GizDeviceNormal	普通设备
1	GizDeviceCenterControl	中控设备
2	GizDeviceSub	子设备

【GizThirdAccountType】

功能描述：第三方账号类型。

枚举 ID	枚举定义	描述
0	GizThirdBAIDU	百度账号
1	GizThirdSINA	新浪账号
2	GizThirdQQ	QQ 账号
3	GizThirdWeChat	微信账号
4	GizThirdFacebook	Facebook 账号

枚举 ID	枚举定义	描述
5	GizThirdTwitter	Twitter 账号

【GizUserAccountType】

功能描述：机智云用户类型。

枚举 ID	枚举定义	描述
0	GizUserNormal	普通用户
1	GizUserPhone	手机用户
2	GizUserEmail	电子邮箱用户
3	GizUserOther	其他用户类型（包括匿名用户）

【GizWifiDeviceNetStatus】

功能描述：设备网络状态类型。

枚举 ID	枚举定义	描述
0	GizDeviceOffline	离线状态
1	GizDeviceOnline	在线状态
2	GizDeviceControlled	可控状态

【GizWifiGAgentType】

功能描述：模组类型。

枚举 ID	枚举定义	描述

枚举 ID	枚举定义	描述
0	GizGAgentMXCHIP	庆科 3162 模组
1	GizGAgentHF	汉枫模组
2	GizGAgentRTK	睿昱模组
3	GizGAgentWM	联盛德模组
4	GizGAgentESP	乐鑫模组
5	GizGAgentQCA	高通模组
6	GizGAgentTI	TI 模组
7	GizGAgentFSK	语音天下模组
8	GizGAgentMXCHIP3	庆科 V3 配置
9	GizGAgentBL	古北模组
10	GizGAgentAtmelEE	Atmel 模组
11	GizGAgentOther	其他模组

【GizUserGenderType】

功能描述：用户性别。

枚举 ID	枚举定义	描述
0	GizUserGenderMale	男
1	GizUserGenderFemale	女

枚举 ID	枚举定义	描述
2	GizUserGenderUnknown	其他

【GizDeviceSharingUserRole】

功能描述：不同的用户角色设备分享时具有不同的设备绑定权限

枚举 ID	枚举定义	描述
0	GizDeviceSharingNormal	普通绑定用户
1	GizDeviceSharingSpecial	潜在 Owner 用户（最早绑定了设备还未进行分享的账号）
2	GizDeviceSharingOwner	Owner 用户
3	GizDeviceSharingGuest	Guest 用户

【GizDeviceSharingType】

功能描述：设备分享类型

枚举 ID	枚举定义	描述
0	GizDeviceSharingByMe	自己发出的分享邀请
1	GizDeviceSharingToMe	分享给自己的邀请

【GizDeviceSharingWay】

功能描述：分享方式

枚举 ID	枚举定义	描述

枚举 ID	枚举定义	描述
0	GizDeviceSharingByNormal	账号分享
1	GizDeviceSharingByQRCode	二维码分享

【GizDeviceSharingStatus】

功能描述：设备分享状态

枚举 ID	枚举定义	描述
0	GizDeviceSharingNotAccepted	未接受
1	GizDeviceSharingAccepted	已接受
2	GizDeviceSharingRefused	已拒绝
3	GizDeviceSharingCancelled	已取消

【GizMessageType】

功能描述：消息类型

枚举 ID	枚举定义	描述
0	GizMessageSystem	系统消息
1	GizMessageSharing	分享消息

【GizMessageStatus】

功能描述：消息状态

枚举 ID	枚举定义	描述

枚举 ID	枚举定义	描述
0	GizMessageUnread	未读消息
1	GizMessageRead	已读消息
2	GizMessageDeleted	已删除消息

【GizOTAFirmwareType】

功能描述：固件升级类型

枚举 ID	枚举定义	描述
0	GizOTAFirmareModule	设备的模组固件
1	GizOTAFirmareMcu	设备的 mcu 固件

【GizSceneItemType】

功能描述：场景项类型

枚举 ID	枚举定义	描述
0	GizSceneItemDevice	设备场景项
1	GizSceneItemGroup	组场景项
2	GizSceneItemDelay	延时场景项

【GizSchedulerType】

功能描述：定时任务类型

枚举 ID	枚举定义	描述

枚举 ID	枚举定义	描述
0	GizSchedulerDelay	延时任务
1	GizSchedulerOneTime	一次性定时任务
2	GizSchedulerWeekRepeat	按周重复定时任务
3	GizSchedulerDayRepeat	按天重复定时任务

【GizScheduleStatus】

功能描述：定时任务执行状态

枚举 ID	枚举定义	描述
0	GizScheduleSucceed	执行成功
1	GizScheduleFailed	执行失败
2	GizScheduleNotDone	执行中

【GizScheduleWeekday】

功能描述：按周重复

枚举 ID	枚举定义	描述
0	GizScheduleSunday	周日
1	GizScheduleMonday	周一
2	GizScheduleTuesday	周二
3	GizScheduleWednesday	周三

枚举 ID	枚举定义	描述
4	GizScheduleThursday	周四
5	GizScheduleFriday	周五
6	GizScheduleSaturday	周六

【GizWifErrorCode】

功能描述：错误码定义。

枚举 ID	枚举定义	描述
0	GIZ_SDK_SUCCESS	Client 发出的请求执行成功
8001	GIZ_SDK_PARAM_FORM_INVALID	Client 发给 Daemon 的 json 格式错误
8002	GIZ_SDK_CLIENT_NOT_AUTHEN	Client 与 Daemon 之间如果没有通过握手认证，任何数据交互都无效
8003	GIZ_SDK_CLIENT_VERSION_INVALID	Client 版本号无效
8004	GIZ_SDK_UDP_PORT_BIND_FAILED	udp 端口绑定失败
8005	GIZ_SDK_DAEMON_EXCEPTION	Daemon 系统错误
8006	GIZ_SDK_PARAM_INVALID	Client 发出的数据请求，Json 格式正确，但参数无效；APP 传入参数无效
8007	GIZ_SDK_APPID_LENGTH_ERROR	appid 长度错误
8008	GIZ_SDK_LOG_PATH_INVALID	日志路径无效
8009	GIZ_SDK_LOG_LEVEL_INVALID	日志级别无效
8010	GIZ_SDK_UID_INVALID	uid 参数无效
8011	GIZ_SDK_TOKEN_INVALID	token 参数无效
8012	GIZ_SDK_USER_NOT_LOGIN	用户未登录
8013	GIZ_SDK_APPID_INVALID	AppID 无效
8014	GIZ_SDK_APP_SECRET_INVALID	App secret 无效
8015	GIZ_SDK_PRODUCT_KEY_INVALID	产品标识无效
8016	GIZ_SDK_PRODUCT_SECRET_INVALID	产品秘钥无效
8017	GIZ_SDK_DEVICE_NOT_IN_LAN	设备不在局域网
8018	GIZ_SDK_PRODUCTKEY_NOT_IN_SPECIAL_LIST	设备不在指定的产品标识列表内

枚举 ID	枚举定义	描述
8019	GIZ_SDK_PRODUCTKEY_NOT_RELATED_WITH_APPID	设备跟当前应用标识未关联
8020	GIZ_SDK_NO_AVAILABLE_DEVICE	批量设置设备域名信息时没有可用设备
8021	GIZ_SDK_DEVICE_CONFIG_SEND_FAILED	设备配置信息发送失败
8022	GIZ_SDK_DEVICE_CONFIG_IS_RUNNING	设备正在配置
8023	GIZ_SDK_DEVICE_CONFIG_TIMEOUT	设备配置超时
8024	GIZ_SDK_DEVICE_DID_INVALID	设备 did 无效
8025	GIZ_SDK_DEVICE_MAC_INVALID	设备 mac 无效
8026	GIZ_SDK_SUBDEVICE_DID_INVALID	子设备 did 无效
8027	GIZ_SDK_DEVICE_PASSCODE_INVALID	设备 passcode 无效
8028	GIZ_SDK_DEVICE_NOT_CENTERCONTROL	不是中控设备
8029	GIZ_SDK_DEVICE_NOT_SUBSCRIBED	设备未订阅
8030	GIZ_SDK_DEVICE_NO_RESPONSE	设备未响应
8031	GIZ_SDK_DEVICE_NOT_READY	设备未就绪
8032	GIZ_SDK_DEVICE_NOT_BINDED	设备未绑定
8033	GIZ_SDK_DEVICE_CONTROL_WITH_INVALID_COMMAND	设备控制指令中包含无效指令
8034	GIZ_SDK_DEVICE_CONTROL_FAILED	设备控制指令执行失败
8035	GIZ_SDK_DEVICE_GET_STATUS_FAILED	设备状态查询失败
8036	GIZ_SDK_DEVICE_CONTROL_VALUE_TYPE_ERROR	设备控制指令参数类型错误
8037	GIZ_SDK_DEVICE_CONTROL_VALUE_OUT_OF_RANGE	设备控制指令参数值不在有效范围内
8038	GIZ_SDK_DEVICE_CONTROL_NOT_WRITABLE_COMMAND	设备控制指令中包含不可写指令
8039	GIZ_SDK_BIND_DEVICE_FAILED	设备绑定失败
8040	GIZ_SDK_UNBIND_DEVICE_FAILED	设备解绑失败
8041	GIZ_SDK_DNS_FAILED	域名解析失败
8042	GIZ_SDK_M2M_CONNECTION_SUCCESS	m2m 连接成功
8043	GIZ_SDK_SET_SOCKET_NON_BLOCK_FAILED	socket 设置非阻塞失败
8044	GIZ_SDK_CONNECTION_TIMEOUT	连接超时
8045	GIZ_SDK_CONNECTION_REFUSED	连接被拒绝
8046	GIZ_SDK_CONNECTION_ERROR	连接错误
8047	GIZ_SDK_CONNECTION_CLOSED	连接被关闭
8048	GIZ_SDK_SSL_HANDSHAKE_FAILED	ssl 握手失败

枚举 ID	枚举定义	描述
8049	GIZ_SDK_DEVICE_LOGIN_VERIFY_FAILED	设备登录验证失败
8050	GIZ_SDK_INTERNET_NOT_REACHABLE	当前外网不可达
8051	GIZ_SDK_M2M_CONNECTION_FAILED	M2M 服务器连接失败
8095	GIZ_SDK_HTTP_SERVER_NOT_SUPPORT_API	HTTP 服务不支持此 API
8096	GIZ_SDK_HTTP_ANSWER_FORMAT_ERROR	openapi 应答格式错
8097	GIZ_SDK_HTTP_ANSWER_PARAM_ERROR	http 应答参数错误
8098	GIZ_SDK_HTTP_SERVER_NO_ANSWER	http 服务无响应
8099	GIZ_SDK_HTTP_REQUEST_FAILED	http 请求失败，比如返回 404 等
8100	GIZ_SDK_OTHERWISE	其他错误
8101	GIZ_SDK_MEMORY_MALLOC_FAILED	Daemon 内存分配失败
8102	GIZ_SDK_THREAD_CREATE_FAILED	Daemon 内部线程创建失败
8120	GIZ_SDK_SCHEDULER_CREATE_FAILED	定时任务创建失败
8121	GIZ_SDK_SCHEDULER_DELETE_FAILED	定时任务删除失败
8122	GIZ_SDK_SCHEDULER_EDIT_FAILED	定时任务信息编辑失败
8123	GIZ_SDK_SCHEDULER_LIST_UPDATE_FAILED	定时任务列表更新失败
8124	GIZ_SDK_SCHEDULER_TASK_EDIT_FAILED	定时任务的任务项编辑失败
8125	GIZ_SDK_SCHEDULER_TASK_LIST_UPDATE_FAILED	定时任务的任务项列表更新失败
8126	GIZ_SDK_SCHEDULER_ID_INVALID	定时任务 ID 无效
8127	GIZ_SDK_SCHEDULER_ENABLE_DISABLE_FAILED	定时任务开启或关闭失败
8128	GIZ_SDK_SCHEDULER_STATUS_UPDATE_FAILED	定时任务状态更新失败
8140	GIZ_SDK_SUBDEVICE_ADD_FAILED	子设备添加失败
8141	GIZ_SDK_SUBDEVICE_DELETE_FAILED	子设备删除失败
8142	GIZ_SDK_SUBDEVICE_LIST_UPDATE_FAILED	子设备列表更新失败
8150	GIZ_SDK_GROUP_ID_INVALID	组 ID 无效
8151	GIZ_SDK_GROUP_PRODUCTKEY_INVALID	组类型无效
8152	GIZ_SDK_GROUP_FAILED_DELETE_DEVICE	删除组设备失败
8153	GIZ_SDK_GROUP_FAILED_ADD_DEVICE	添加组设备失败
8154	GIZ_SDK_GROUP_GET_DEVICE_FAILED	组设备列表更新失败
8155	GIZ_SDK_GROUP_CREATE_FAILED	添加组失败
8156	GIZ_SDK_GROUP_DELETE_FAILED	删除组失败
8157	GIZ_SDK_GROUP_EDIT_FAILED	编辑组信息失败

枚举 ID	枚举定义	描述
8158	GIZ_SDK_GROUP_LIST_UPDATE_FAILED	组列表更新失败
8159	GIZ_SDK_GROUP_COMMAND_WRITE_FAILED	组操作执行失败
8170	GIZ_SDK_SCENE_CREATE_FAILED	场景添加失败
8171	GIZ_SDK_SCENE_DELETE_FAILED	场景删除失败
8172	GIZ_SDK_SCENE_EDIT_FAILED	场景信息编辑失败
8173	GIZ_SDK_SCENE_LIST_UPDATE_FAILED	场景列表更新失败
8174	GIZ_SDK_SCENE_ITEM_LIST_EDIT_FAILED	场景项列表编辑失败
8175	GIZ_SDK_SCENE_ITEM_LIST_UPDATE_FAILED	场景项列表更新失败
8176	GIZ_SDK_SCENE_ID_INVALID	场景 ID 无效
8177	GIZ_SDK_SCENE_RUN_FAILED	场景执行失败
8178	GIZ_SDK_SCENE_STATUS_UPDATE_FAILED	场景状态更新失败
8201	GIZ_SDK_DATAPOINT_NOT_DOWNLOAD	配置文件还未下载
8202	GIZ_SDK_DATAPOINT_SERVICE_UNAVAILABLE	配置文件服务不可用
8203	GIZ_SDK_DATAPOINT_PARSE_FAILED	配置文件解析失败
8300	GIZ_SDK_SDK_NOT_INITIALIZED	SDK 未初始化
8301	GIZ_SDK_APK_CONTEXT_IS_NULL	Context 无效，无法启动
8302	GIZ_SDK_APK_PERMISSION_NOT_SET	app 权限不足
8303	GIZ_SDK_CHMOD_DAEMON_REFUSED	无法修改 daemon 的执行权限
8304	GIZ_SDK_EXEC_DAEMON_FAILED	daemon 程序执行失败
8305	GIZ_SDK_EXEC_CATCH_EXCEPTION	尝试运行 daemon 时发生异常
8306	GIZ_SDK_APPID_IS_EMPTY	APPID 为空
8307	GIZ_SDK_UNSUPPORTED_API	不支持的 API
8308	GIZ_SDK_REQUEST_TIMEOUT	Client 如果等不到 Daemon 的回复，就向 APP 返回操作超时
8309	GIZ_SDK_DAEMON_VERSION_INVALID	Daemon 版本号无效
8310	GIZ_SDK_PHONE_NOT_CONNECT_TO_SOFTAP_SSID	手机没有连接软 AP 热点
8311	GIZ_SDK_DEVICE_CONFIG_SSID_NOT_MATCHED	手机热点和要配置的路由 ssid 不匹配
8312	GIZ_SDK_NOT_IN_SOFTAPMODE	设备不在 softap 模式
8313	GIZ_SDK_CONFIG_NO_AVAILABLE_WIFI	设备配置时无可用 wifi
8314	GIZ_SDK_RAW_DATA_TRANSMIT	设备上报透传数据的标识
8315	GIZ_SDK_PRODUCT_IS_DOWNLOADING	正在下载设备的产品定义
8316	GIZ_SDK_START_SUCCESS	SDK 启动成功

枚举 ID	枚举定义	描述
8317	GIZ_SDK_NEED_UPDATE_TO_LATEST	SDK 需要升级到最新版本
8350	GIZ_SDK_OTA_FIRMWARE_IS_LATEST	当前固件是最新版本，不需要升级
8351	GIZ_SDK_OTA_FIRMWARE_CHECK_UPDATE_FAILED	固件检查更新失败
8352	GIZ_SDK_OTA_FIRMWARE_DOWNLOAD_OK	固件下载成功
8353	GIZ_SDK_OTA_FIRMWARE_DOWNLOAD_FAILED	固件下载失败
8354	GIZ_SDK_OTA_DEVICE_BUSY_IN_UPGRADE	设备忙，固件正在升级
8355	GIZ_SDK_OTA_PUSH_FAILED	固件推送失败
8356	GIZ_SDK_OTA_FIRMWARE_VERSION_TOO_LOW	固件版本过低
8357	GIZ_SDK_OTA_FIRMWARE_CHECK_FAILED	固件校验失败
8358	GIZ_SDK_OTA_UPGRADE_FAILED	固件升级失败
8359	GIZ_SDK_OTA_FIRMWARE_VERIFY_SUCCESS	固件校验成功
8360	GIZ_SDK_OTA_DEVICE_NOT_SUPPORT	设备不支持手机 OTA 升级
9001	GIZ_OPENAPI_MAC_ALREADY_REGISTERED	mac already registered!
9002	GIZ_OPENAPI_PRODUCT_KEY_INVALID	product_key invalid
9003	GIZ_OPENAPI_APPID_INVALID	appid invalid
9004	GIZ_OPENAPI_TOKEN_INVALID	token invalid
9005	GIZ_OPENAPI_USER_NOT_EXIST	user not exist
9006	GIZ_OPENAPI_TOKEN_EXPIRED	token expired
9007	GIZ_OPENAPI_M2M_ID_INVALID	m2m_id invalid
9008	GIZ_OPENAPI_SERVER_ERROR	server error
9009	GIZ_OPENAPI_CODE_EXPIRED	code expired
9010	GIZ_OPENAPI_CODE_INVALID	code invalid
9011	GIZ_OPENAPI_SANDBOX_SCALE_QUOTA_EXHAUSTED	sandbox scale quota exhausted!
9012	GIZ_OPENAPI_PRODUCTION_SCALE_QUOTA_EXHAUSTED	production scale quota exhausted!
9013	GIZ_OPENAPI_PRODUCT_HAS_NO_REQUEST_SCALE	product has no request scale!
9014	GIZ_OPENAPI_DEVICE_NOT_FOUND	device not found!
9015	GIZ_OPENAPI_FORM_INVALID	form invalid!
9016	GIZ_OPENAPI_DID_PASSCODE_INVALID	did or passcode invalid!
9017	GIZ_OPENAPI_DEVICE_NOT_BOUND	device not bound!
9018	GIZ_OPENAPI_PHONE_UNAVALIABLE	phone unavailable!

枚举 ID	枚举定义	描述
9019	GIZ_OPENAPI_USERNAME_UNAVALIABLE	username unavailable!
9020	GIZ_OPENAPI_USERNAME_PASSWORD_ERROR	username or password error!
9021	GIZ_OPENAPI_SEND_COMMAND_FAILED	send command failed!
9022	GIZ_OPENAPI_EMAIL_UNAVALIABLE	email unavailable!
9023	GIZ_OPENAPI_DEVICE_DISABLED	device is disabled!
9024	GIZ_OPENAPI_FAILED_NOTIFY_M2M	fail to notify m2m!
9025	GIZ_OPENAPI_ATTR_INVALID	attr invalid!
9026	GIZ_OPENAPI_USER_INVALID	user invalid!
9027	GIZ_OPENAPI_FIRMWARE_NOT_FOUND	firmware not found!
9028	GIZ_OPENAPI_JD_PRODUCT_NOT_FOUND	JD product info not found!
9029	GIZ_OPENAPI_DATAPOINT_DATA_NOT_FOUND	datapoint data not found!
9030	GIZ_OPENAPI_SCHEDULER_NOT_FOUND	scheduler not found!
9031	GIZ_OPENAPI_QQ_OAUTH_KEY_INVALID	qq oauth key invalid!
9032	GIZ_OPENAPI_OTA_SERVICE_OK_BUT_IN_IDLE	ota upgrade service OK, but in idle or disable!
9033	GIZ_OPENAPI_BT_FIRMWARE_UNVERIFIED	bt firmware unverified, except verify device!
9034	GIZ_OPENAPI_BT_FIRMWARE_NOTHING_TO_UPGRADE	bt firmware is OK, but nothing to upgrade!
9035	GIZ_OPENAPI_SAVE_KAIROSDB_ERROR	Save kairosdb error!
9036	GIZ_OPENAPI_EVENT_NOT_DEFINED	event not defined!
9037	GIZ_OPENAPI_SEND_SMS_FAILED	send sms failed!
9038	GIZ_OPENAPI_APPLICATION_AUTH_INVALID	X-Gizwits-Application-Auth invalid!
9039	GIZ_OPENAPI_NOT_ALLOWED_CALL_API	Not allowed to call deprecated API!
9040	GIZ_OPENAPI_BAD_QRCODE_CONTENT	bad qrcode content!
9041	GIZ_OPENAPI_REQUEST_THROTTLED	request was throttled
9042	GIZ_OPENAPI_DEVICE_OFFLINE	device offline!
9043	GIZ_OPENAPI_TIMESTAMP_INVALID	'X-Gizwits-Timestamp invalid!
9044	GIZ_OPENAPI_SIGNATURE_INVALID	X-Gizwits-Signature invalid!
9045	GIZ_OPENAPI_DEPRECATED_API	API deprecated!
9046	GIZ_OPENAPI_REGISTER_IS_BUSY	Register already in progress!
9065	GIZ_OPENAPI_APPID_PK_NOT_RELATION	appid has no relation with pk!

枚举 ID	枚举定义	描述
9066	GIZ_OPENAPI_CALL_INNER_FAILED	call innerapi failed!
9068	GIZ_OPENAPI_DEVICE_SHARING_NOT_ENABLED	Device share not enabled for this product!
9069	GIZ_OPENAPI_NOT_FIRST_USER_OF_DEVICE	You are not the first user of this device!
9072	GIZ_OPENAPI_PRODUCT_KEY_AUTHEN_FAULT	App auth key invalid!
9073	GIZ_OPENAPI_BUSY_NOW	operation in process, please try again later.
9074	GIZ_OPENAPI_TWITTER_CONSUMER_KEY_INVALID	App twitter consumer_key or consumer_secret not valid.
9080	GIZ_OPENAPI_CANNOT_SHARE_TO_SELF	can not share device to self!
9081	GIZ_OPENAPI_ONLY_OWNER_CAN_SHARE	guest or normal user can not share device!
9082	GIZ_OPENAPI_NOT_FOUND_GUEST	guest user not found!
9083	GIZ_OPENAPI_GUEST_ALREADY_BOUND	guest user alread bound!
9084	GIZ_OPENAPI_NOT_FOUND_SHARING_INFO	sharing record not found!
9085	GIZ_OPENAPI_NOT_FOUND_THE_MESSAGE	message record not found!
9087	GIZ_OPENAPI_SHARING_IS_WAITING_FOR_ACCEPT	sharing alread created, waiting for the guest to accept!
9088	GIZ_OPENAPI_SHARING_IS_EXPIRED	sharing record expired!
9089	GIZ_OPENAPI_SHARING_IS_COMPLETED	sharing record status is not unaccept!
9090	GIZ_OPENAPI_INVALID_SHARING_BECAUSE_UNBINDING	owner binding disabled!
9092	GIZ_OPENAPI_ONLY_OWNER_CAN_BIND	owner exist, guest can not bind!
9093	GIZ_OPENAPI_ONLY_OWNER_CAN_OPERATE	permission denied, you are not owner!
9094	GIZ_OPENAPI_SHARING_ALREADY_CANCELLED	sharing already canceled!
9095	GIZ_OPENAPI_OWNER_CANNOT_UNBIND_SELF	can not unbind self!
9096	GIZ_OPENAPI_ONLY_GUEST_CAN_CHECK_QRCODE	permission denied, you are not guest!
9098	GIZ_OPENAPI_MESSAGE_ALREADY_DELETED	notify delele binding failed!
9099	GIZ_OPENAPI_BINDING_NOTIFY_FAILED	notify delele binding failed!
9100	GIZ_OPENAPI_ONLY_SELF_CAN_MODIFY_ALIAS	permission denied, you are not owner or guest!
9101	GIZ_OPENAPI_ONLY_RECEIVER_CAN_MARK_MESSAGE	permission denied, you are not the receiver!
9102	GIZ_OPENAPI_GUEST_NOT_BIND	guest not bind
9103	GIZ_OPENAPI_CANNOT_TRANSFER_OWNER_TO_SELF	can not transfer owner privilege to self!
9104	GIZ_OPENAPI_TRANSFER_OWNER_TO_LIMIT_GUEST	can not transfer owner privilege to a time

枚举 ID	枚举定义	描述
		limited guest!
9999	GIZ_OPENAPI_RESERVED	reserved