Android SDK 快速接入文档

目录

概要

青莲云作为物联网后端云服务，用户可以使用青莲云提供的 Android SDK 快速进行 App 开发。

1、SDK 介绍

```
iotcloud_sdk_x.x.x
├── arm64-v8a
│   └── libcloudcore.so         动态链接库
├── armeabi
│   └── libcloudcore.so         动态链接库
├── armeabi-v7a
│   └── libcloudcore.so         动态链接库
├── iot_cloud_sdk_x.x.x.jar     SDK 核心库文件
├── mips
│   └── libcloudcore.so         动态链接库
├── mips64
│   └── libcloudcore.so         动态链接库
├── x86
│   └── libcloudcore.so         动态链接库
└── x86_64
    └── libcloudcore.so         动态链接库
```

2、AndroidMinfest.xml 配置

```
    <!-- sdcard -->

    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />









    <!-- 联网 -->

    <uses-permission android:name="android.permission.INTERNET"
/>

    <uses-permission
android:name="android.permission.CHANGE_NETWORK_STATE" />

    <uses-permission
android:name="android.permission.CHANGE_WIFI_STATE" />

    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />

    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />

    <uses-permission
android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"
/>

    <uses-permission
android:name="android.permission.WAKE_LOCK" />



    <!-- 定位 -->
```

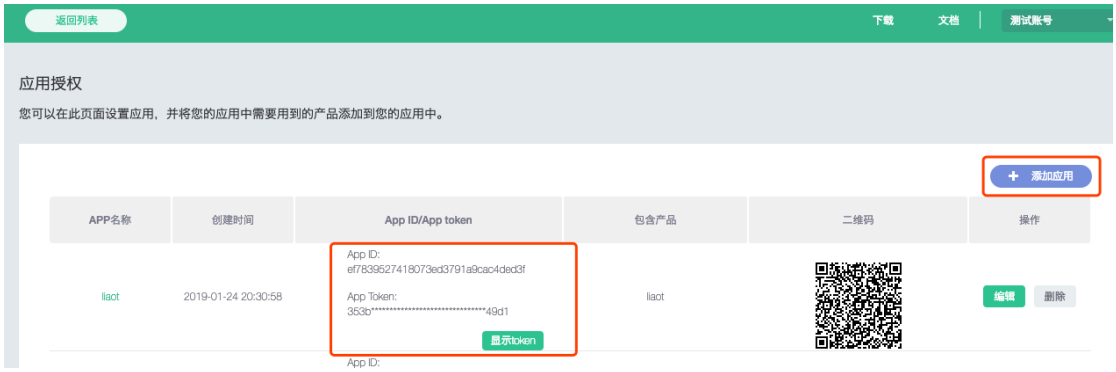
```
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />



    <!-- SDK 使用的签名级别的权限 -->

    <permission

        android:name="应用包名.permission.IOTCLOUD_MESSAGE"

        android:protectionLevel="signature" />

    <uses-permission android:name="应用包
名.permission.IOTCLOUD_MESSAGE" />




    <!-- 消息服务 -->

    <service

            android:name="com.iot.cloud.sdk.api.MQTTService"

            android:enabled="true"

            android:exported="false"

            android:permission="应用包
名.permission.IOTCLOUD_MESSAGE" />


    <!-- 自定义广播接收者，接收消息和指令 -->

    <receiver

            android:name="广播接收者全限定类名"

            android:enabled="true"

            android:exported="false"
```

```
            android:permission="应用包
名.permission.IOTCLOUD_MESSAGE">

            <intent-filter>

                <action
android:name="com.iot.cloud.sdk.intent.MESSAGE_RECEIVED" />

            </intent-filter>

    </receiver>
```

3、SDK 初始化

1、在官网左上角点击授权管理

2、添加应用即可获得 AppID 和 AppToken



推荐在 Application 中进行初始化，注意只需要在主进程初始化一次即可

```
@Override
public void onCreate() {
    super.onCreate();
    // 判断是主进程
    if (getApplicationInfo().packageName
```

```
    .equals(getCurProcessName(getApplicationContext()))) {
        /**
            * 初始化 SDK
            */
        IotCloudSDK.init(this, AppId, AppToken);

        // 设置需要登录时的状态回调，用来处理页面跳转，比如跳转到登录
页面
        IotCloudSDK.setNeedLoginListener
(new INeedLoginListener() {
            @Override
            public void onNeedLogin(Context context) {
                try {
                    Intent intent = new Intent(context,
                     SetAppIdActivity.class);

intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);

intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(intent);
                } catch (Exception e) {
                    Intent intent = new Intent(context,
SetAppIdActivity.class);

intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(intent);
                }
```

```
            }
        });

        DeviceController.bindGlobalMessage(this);

        // 返回当前 SDK 的长连接状态
        IotCloudSDK.setConnectionStatusListener(new
IConnectionStatusListener() {
            @Override
            public void onConnectionStatus(int i) {
                LogUtils.e("MQTT 当前状态: " + i);
            }
        });

    }
}
public static String getCurProcessName(Context context) {

    int pid = android.os.Process.myPid();

    ActivityManager activityManager = (ActivityManager) context
            .getSystemService(Context.ACTIVITY_SERVICE);
    for (ActivityManager.RunningAppProcessInfo appProcess :
activityManager
            .getRunningAppProcesses()) {
```

```
        if (appProcess.pid == pid) {
            return appProcess.processName;
        }
    }
    return null;
}

@Override
public void onMessage(CloudMessage cloudMessage) {
    if (cloudMessage.getErrorMessage().getCode() ==
CommonError.MQTT_MESSAGE_SUCCESS) {
        // success
        switch (cloudMessage.getType()) {
            case CloudMessageType.TYPE_OTA_REV: // 有设备需要手动
OTA 升级

                break;
            case CloudMessageType.TYPE_SHARE_REV: // 有分享消息，
请到分享消息列表查看

                int shareType =
CloudShareMessageType.getShareType(cloudMessage);
                if (shareType == CloudShareMessageType.TYPE_DEVICE)
{
                    // 设备分享消息
                    LogUtils.e("设备分享消息");
                } else if (shareType ==
CloudShareMessageType.TYPE_FAMILY) {
```

```
                    // 家庭分享消息
                    LogUtils.e("家庭分享消息");
                }


                );


                break;
            }
        } else {
            // 错误消息
            LogUtils.e(cloudMessage.getErrorMessage());
        }
}
```

4、发送手机验证码

```
// Zone.CN 表示手机区号，Zone 是 SDK 中的类
IotCloudSDK.getUserManager().getRegisterSMSCode("手机号",
Zone.CN, new ISDKCallback() {
    @Override
    public void onSuccess() {
        // 成功
    }


    @Override
    public void onError(ErrorMessage errorMessage) {
```

```
        // 失败，查看错误码和错误信息
        // errorMessage.getCode(), errorMessage.getMessage()
    }
});
```

5、注册

```
// 注册接口
// Zone.CN 表示手机区号，Zone 是 SDK 中的类
IotCloudSDK.getUserManager().userRegister("手机号", "密码", "短信验证码", Zone.CN, new ISDKCallback() {
    @Override
    public void onSuccess() {
        // 成功
    }

    @Override
    public void onError(ErrorMessage errorMessage) {
        // 失败，查看错误码和错误信息
        // errorMessage.getCode(), errorMessage.getMessage()
    }
});
```

6、登录

```
// 登录接口
```

```
// Zone.CN 表示手机区号，Zone 是 SDK 中的类

IotCloudSDK.getUserManager().userLogin("手机号", "密码", Zone.CN,
new ISDKCallback() {

    @Override

    public void onSuccess() {

        // 成功，可以调用其他接口使用 SDK 了

    }


    @Override

    public void onError(ErrorMessage errorMessage) {

        // 失败，查看错误码和错误信息

        // errorMessage.getCode(), errorMessage.getMessage()

    }

});
```

7、获取产品列表

1、添加产品配置到应用中就会在产品列表接口中获取到


```
IotCloudSDK.getDeviceManager().getProductList(new
ICallback<List<Product>>() {

    @Override
```

```
    public void onSuccess(List<Product> products) {
        // 获取产品列表
        // Product.productId 表示产品 id
    }

    @Override
    public void onError(ErrorMessage errorMessage) {

    }
});
```

8、获取设备列表

```
IotCloudSDK.getDeviceManager().getDeviceList(new
ICallback<List<CloudDevice>>() {
    @Override
    public void onSuccess(List<CloudDevice> deviceList) {
        // 成功
    }

    @Override
    public void onError(ErrorMessage errorMessage) {
        // 失败，查看错误码和错误信息
        // errorMessage.getCode(), errorMessage.getMessage()
    }
});
```

9、发现设备

```
// 注意，目前相关模组仅支持 2.4GWIFI，5GWIFI 和混合 WIFI 可能无法正常工作

// WIFI 模组目前支持、乐鑫 ESP8266、汉枫、庆科 Mico、高通等

IotCloudSDK.getDeviceManager().startEspSmart("路由器名称", "路由器密码", 产品 id, new DeviceFoundListener() {

    @Override

    public void onError(ErrorMessage errorMessage) {

        // 一般不会支持此回调

    }


    @Override

    public void findDevice(CloudDevice device) {

        // 可能会多次回调，回调在主线程

        // 回调了设备信息，其中 iotId 和 iotToken 可用于后续绑定设备

        // iotId 表示设备唯一 id

        // iotToken 主要用于业务上的信息验证，一般不要展示该字段

        String iotId = device.getIotId();

        String iotToken = device.getIotToken();

    }


    @Override

    public void noFindDevice() {

        // 在没有发现设备信息的时候或者超时后总会回调此接口

        // 可以表示本次调用完成，不代表没有发现设备
```

```
    }

});
```

10、绑定设备

```
// 绑定接口，传入设备对象

IotCloudSDK.getDeviceManager().bindDevice(CloudDevice
cloudDevice,

    new IBindDeviceCallback() {

        @Override

        public void onSuccess(BindDeviceInfo bindDeviceInfo) {

            // 绑定成功

        }



        @Override

        public void onError(ErrorMessage errorMessage,
BindDeviceInfo bindDeviceInfo) {

            // 绑定失败

        }

});
```

11、控制设备

1、控制设备之前需要为该设备所属的产品增加功能点，点击进入产品功能后，添加功能点即可。


```
// 从产品列表，设备列表获取的元素

// Product 的 productId

// CloudDevice.getIotId

// CloudDevice.getSubIotId  该字段可能为空，当 CloudDevice 实例是子设备的时候该字段有值

// CloudDevice.isSubDevice  判断是否是子设备

DeviceController deviceController = new DeviceController(产品 id, "设备 id", "子设备 id 如果有");


MessageData.Builder builder = new MessageData.Builder();


// key 是云端产品功能页面的数据点功能键

// value 可以是 二进制数据类型、int、float、String

// put 方法可以多次调用，同一个 key 不会覆盖，会添加多次

builder.put("key", value);

builder.put("key", value);

builder.put("key", value);


// 发送命令
```

```
deviceController.sendCommand(builder, new ISDKCallback() {
    @Override
    public void onSuccess() {
        // 发送成功
    }

    @Override
    public void onError(ErrorMessage errorMessage) {
        // 发送失败
    }
});
```

12、接收设备上报消息

```
deviceController.setDPUPdateListener(new
DeviceController.DPUpdateListener() {
    @Override
    public void onDPUpdate(List<MessageData> list) {
        // list 可能为空
        // MessageData.key 表示数据点功能键
        // 根据 key 来判断是哪个数据点，然后使用对应的 getValue 方法，
强转类型即可
        // getIntValue、getFloatValue、getStringValue、getByteArray
    }
});
```