

蜻蜓 OPEN SDK Quick Start

错误码	错误消息	解释
0	Success	成功返回结果
10000	Internal Error	服务器内部错误
10001	Channel Access Not Allow	对该专辑无权限访问
10002	Limit Excced	访问频次超过限制
10003	Auth Failure	认证失败
10004	Item already purchased	重复购买的错误消息
10005	User Not Exist	用户不存在
10006	User Not Login	用户未登录
10007	Channel Not Exist	该专辑不存在
10008	Empty Device ID	缺少device Id参数
10009	Invalid page or pagesize	页码或页容量非法
10010	Already refunded	商品已退款，勿重复退款
10011	Invalid Param	参数错误
10012	Get Audio Resource Failed	获取音频资源失败
10013	Audio not purchased by user	该音频未购买
10014	Not Purchased, return free audio	音频未购买，返回免费试听节目
10015	No Free Audio	无试听节目
10016	Transcoding, please retry later	音频转码中，请稍后重试
20000	No NetWork	网络异常
20001	Miss Auth Request	授权请求缺失
20002	Miss Auth RedirectUrl	授权回调地址缺失
20003	Refresh Token fail	token刷新失败

- 1. 初始化
 - 1.1 gradle配置
 - 1.1.1 sdk依赖
 - 1.1.2 库版本冲突和jdk8兼容
 - 1.2 初始化代码
 - 1.3 需求权限
 - 1.4 混淆
- 2. 授权

- 2.1 授权启动
 - 2.2 授权回调
 - 2.3 授权移除
 - 2.4 第三方账号登陆
- 3. 用户信息
- 4. 支付
- 5. 播放器
 - 5.1 概述
 - 5.2 使用步骤
 - 5.2.1 连接播放器
 - 5.2.2 播放资源准备
 - 5.2.3 播放监听
 - 5.2.4 播放控制
 - 5.2.5 播放器暴露的其它控制方法
- 6. 常用接口

初始化

gradle配置

建议使用compileSdkVersion 27及以上

sdk依赖

引用蜻蜓sdk的模块的build.gradle配置。

- maven依赖

```
repositories {
    maven {
        url uri('http://maven.qingting.fm/')
    }
}
dependencies {
    implementation 'fm.qingting.open.android:qtsdk:1.3.0' //具体请参考网站集成最新版本
}
```

库版本冲突和jdk8兼容

为避免库版本冲突和jdk8的不兼容，请在gradle中设置以下信息

```
configurations.all {
    resolutionStrategy.eachDependency { DependencyResolveDetails
details ->
        def requested = details.requested
        switch (requested.group) {
```

```

        // 统一android support库版本 (multidex除外) , 防止冲突
        case 'com.android.support':
            if (!requested.name.startsWith("multidex")) {
                details.useVersion '27.0.2'
            }
            break
        // 统一okhttp库版本, 防止冲突
        case 'com.squareup.okhttp3':
            details.useVersion '3.10.0'
            break
        // 统一gson库版本, 防止冲突
        case 'com.google.code.gson':
            if (requested.name.startsWith("gson")) {
                details.useVersion '2.8.0'
            }
            break
    }
}

compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}

```

初始化代码

首先通过蜻蜓官方渠道获取用于初始化SDK的 Client Id并上填写package name 以及签名文件的sha1值。。然后新建一个名字叫做 MyQTApplication Java Class ,让它继承自 Application 类, 代码如下:

```

public class MyQTApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public void onCreate() {
        super.onCreate();
        QTSdk.setHost("服务器地址");
        // 初始化参数依次为 this, ClientId
        QTSdk.init("TEST_CLIEND_ID");
        QTSdk.setAuthRedirectUrl("http://qttest.qingting.fm");
    }
}

```

需求权限

打开 AndroidManifest.xml 文件来配置 SDK 所需要的手机的访问权限以及配置 MyQTApplication 类:

```
<!-- 蜻蜓opensdk权限声明 START -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- 权限声明 END -->

<application
    android:name=".MyQTApplication" >

</application>
```

混淆

```
-dontwarn retrofit2.Platform$Java8
```

授权

授权启动

蜻蜓基于OAuth2.0授权模式，如调用者后端需要回调可以通过设置回调地址实现：

```
QTSDK.setAuthRedirectUrl(url);
```

授权方式分为web授权与唤起蜻蜓app授权两种方式，参考示例如下：

```
//启动web授权
QTSDK.startWebAuthorize(getActivity(), new QTAuthCallBack() {
    @Override
    public void onComplete(QTAuthResponse response) {
        //授权成功
    }

    @Override
    public void onException(QTException e) {
        //授权异常，具体查看异常code
    }

    @Override
    public void onCancel() {
        //用户取消授权
    }
});
```

```
//先尝试启动蜻蜓app, 如果失败则自动启动web授权
QTSDK.startAuthorize(getActivity(), new QTAuthCallBack() {
    @Override
    public void onComplete(QTAuthResponse response) {
        //授权成功
    }

    @Override
    public void onException(QTException e) {
        //授权异常, 具体查看异常code
    }

    @Override
    public void onCancel() {
        //用户取消授权
    }
});
```

授权移除

使用者可以调用clear方法清除本地授权.

```
QTSDK.clear();
```

第三方账号登陆

第三方账号可以通过自己的账号token登陆蜻蜓SDK,这里通过后台获取token的时候需要提供蜻蜓的sdk的deviceid{QTSDK.getDeviceId()}给到后端接口,否则有可能无法通过校验

```
UserToken token = new UserToken();
    token.setAccessToken("111111");
    token.setRefreshToken("111111");
    token.setUserId("111111");
    token.setExpiresIn(3600);
    QTSDK.thirdPartLogin(token, new QTAuthCallBack() {
        @Override
        public void onComplete(UserToken response) {
            //授权成功
            Toast.makeText(getBaseContext(), "成功",
                Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onException(QTException e) {
            //授权异常, 具体查看异常code
            Toast.makeText(getBaseContext(), "失败",
                Toast.LENGTH_SHORT).show();
            Logger.e(AuthActivity.class.getSimpleName(), e.getMessage());
        }
    });
```

```
    }

    @Override
    public void onCancel() {
        //用户取消授权
        Toast.makeText(getBaseContext(), "取消授权",
            Toast.LENGTH_SHORT).show();
    }
});
```

用户信息

获取用户信息

```
QTUserCenter.getUserInfo(new QTCallback<UserInfo>() {
    @Override
    public void done(UserInfo result, QTException e) {
        if(result!=null && e!=null){
            已获取
        }else{
            出错或者未登陆
        }
    }
});
```

支付

蜻蜓SDK可以获取的内容分为免费内容与收费内容两部分，其中免费内容无须授权即可直接获取，收费内容则需要用户授权之后购买才可获取。可以调用sdk的startQTPay方法并传入代购买专辑的商品id，节目id(选填)，支付结果回调来启动蜻蜓支付流程。

```
if (!TextUtils.isEmpty(QTUserCenter.getQTUserId())) {
    QTSDK.startQTPay(itemId, programId, (result, e) -> {
        if (result != null) {
            switch (result) {
                case "0":
                    result = "成功";
                    break;
                case "1":
                    result = "失败";
                    break;
                case "2":
                    result = "取消";
                    break;
                case "3":
                    result = "未知";
                    break;
                case "4":
```

```

                result = "已购";
                break;
            }
            Toast.makeText(getBaseContext(), "购买回调状
态:"+result, Toast.LENGTH_SHORT).show();
        } else {
            //处理异常
            Toast.makeText(getBaseContext(), "购买回调异常",
Toast.LENGTH_SHORT).show();
        }
    });
} else {
    Toast.makeText(getBaseContext(), "请先登录",
Toast.LENGTH_SHORT).show();
}

```

播放器

概述

底层播放器的实现已经做了替换，对于`QTPlayer`的具体使用可参考demo中的`PlayerActivity`，关于播放器中标记为弃用状态的只是为了兼容sdk老版本播放器

- 建议不要在使用老版本的播放器接口，后续将放弃维护并逐步移除
- 对于电台播放，因为是基于HLS的（即High Live Stream），每次加载的都是ts片段，因此是不可能拿到完整的durations，每次拿到的都是当前ts片的时间，所以不建议针对电台节目使用播放进度

使用步骤

连接播放器

建议在播放器服务连接成功后，初始化播放页面的播放器

```

/**
 * 获取播放器，通过[callback]的回调来获取播放器
 * - 第一次获取或者在[release]后是一个异步过程，因为这涉及到播放器服务的连接过程
 * - 当不再需要播放器服务的时候，请调用[release]
 * @param callback 播放器连接回调接口
 * - 成功可通过 [Connect2PlayerCallback.onConnected] 回调获取播放器
 * - 断连，通过 [Connect2PlayerCallback.onDisconnected] 回调
 * - 失败或异常，通过 [Connect2PlayerCallback.onFail] 回调
 */
QTPlayerManager.obtainPlayer(object :
QTPlayerManager.Connect2PlayerCallback {
    override fun onConnected(player: QTPlayer) {
        this@PlayerActivity.player = player
    }
}

```

```

        override fun onDisconnected() {
            player = null
        }

        override fun onFair(e: QTException) {
            e.printStackTrace()
            player = null
        }
    })
}

```

不再需要播放器服务，切记调用`QTPlayerManager.release()`，避免资源浪费。需要注意的是，一旦调用`QTPlayerManager.release()`，播放器服务和播放器都将停止并释放（即无法正常播放），除非重新调用`QTPlayerManager.obtainPlayer()`，恢复播放器服务功能。

- 若想要细粒度的调用，仅释放播放器，请使用`QTPlayer.release()`

播放监听

```

val playbackListener = object : QTPlaybackListener() {
    /**
     * url准备失败，对应于老版本的
     [fm.qingting.qtsdk.player.QTPlayer.PlayState.SOURCEFAIL]
     * - 通常发生的原因是获取url失败
     */
    override fun onPrepareUrlFail()

    /**
     * 播放状态变化回调
     * @param playbackState 播放状态
     */
    fun onPlaybackStateChanged(playbackState: PlaybackState)

    /**
     * 播放进度改变回调，时间单位：mills
     * @param currentPositionMS 当前进度
     * @param bufferedPositionMS 当前缓冲进度
     * @param durationMS 总时间
     */
    fun onPlaybackProgressChanged(currentPositionMS: Long,
        bufferedPositionMS: Long, durationMS: Long)

    /**
     * 播放器错误回调，对应于老版本的
     [fm.qingting.qtsdk.player.QTPlayer.PlayState.ERROR]
     */
    fun onPlayerError(error: PlaybackException)

    /**
     * 播放倍速变更回调
     */
    fun onPlaybackSpeedChanged(speed: Float)
}

```



```
}

// 添加播放监听
player.addPlaybackListener(playbackListener)

// 移除播放监听
player.removePlaybackListener(playbackListener)
```

播放资源准备

```
// 专辑节目准备
player.prepare(@NotNull int channelId, int programId);

// 电台准备
player.prepare(@NotNull int channelId);
```

播放控制

```
// 播放/恢复当前播放
player.play();

// 暂停当前播放
player.pause();

// 停止播放
player.stop();

// 快进
player.fastForward();

// 后退
player.rewind();

// 设置媒体资源的播放进度
player.seekTo(progressMillis);
```

播放器暴露的其它控制方法

```
QTPlayer {
    /**
     * 释放播放器，当不再需要播放的时候务必释放
     */
    void release();

    /**
     * 设置播放器的音量，0~1的范围是基于当前设备的当前音量调节的，并不会设置设备硬件音量
```

```
*
* @param volume the volume from 0 to 100
*/
void setVolume(float volume);

/**
 * 获取播放器音量，而非设备音量
 *
 * @return the volume from 0 to 1
 */
float getVolume();

/**
 * 获取播放器状态
 * @return
 */
PlaybackState getPlaybackState();

/**
 * 设置播放倍率
 *
 * @param rate 可用倍率为0.5f, 1f, 1.25f, 1.5f, 2f
 */
void setSpeedRate(float rate);

/**
 * 获取当前播放倍率
 *
 * @return the rate
 */
float getSpeedRate();

/**
 * 获取当前播放进度
 * @return 播放进度，单位为毫秒
 */
long getPositionMS();

/**
 * 获取当前媒体资源的持续时间
 * - 若小于0表示媒体资源为设置持续时间或者没有媒体资源
 * @return 持续时间，单位为毫秒
 */
long getDurationMS();

/**
 * 启动调试
 */
void startDebug();

/**
 * 停止调试
 */
void stopDebug();
```

```
OkHttpClient.Builder newOkHttpClientBuilder();

/**
 * 重置媒体资源http请求客户端
 * - 通过[client]设置免流代理
 */
void resetOkHttpClient(@NonNull OkHttpClient client);
}
```

常用接口描述

`public static void init(Context context, String clientId)`

初始化蜻蜓SDK, 建议在应用程序初始化时调用

- **Parameters:**
 - `context` — the context
 - `clientId` — 合作方的ID

`public static void thirdPartLogin(String thirdPartToken, QTCallback callback)`

授权登陆,如该平台未绑定蜻蜓账号则会触发蜓账号授权页面

- **Parameters:**
 - `thirdPartToken` — 第三方账号平台token
 - `callback` — 结果回调{@link QTCallback}

`public static void clear()`

清除本地登陆态

`public static void requestChannelOndemandList(Integer categoryId, String order, Integer pageIndex, QTCallback<QTListEntity> callback)`

获取点播专辑列表

- **Parameters:**
 - `categoryId` — 分类id 可为空
 - `podcaordersterId` — 可选[bytrend按热度排序, byupdate按专辑更新时间排序], 不填时按默认排序输出
 - `pageIndex` — 分页信息从1开始, 默认第一页, 每页30条
 - `callback` — {@link QTCallback}结果回调,返回类容参考{@link Channel}

`public static void requestChannelOndemandCategories(QTCallback<List> callback)`

获取点播专辑分类

- **Parameters:** `callback` — {@link QTCallback}结果回调,返回类容参考{@link Category}

```
public static void requestChannelOnDemand(int channelId, QTCallback  
callback)
```

获取点播专辑

- **Parameters:**

- **channelId** — 专辑id
- **callback** — {@link QTCallback}结果回调,返回类容参考{@link Channel}

```
public static void requestChannelPrmission(int channelId, QTCallback  
callback)
```

获取点播专辑权限, 回调里会给出当前用户对该专辑的购买情况

- **Parameters:**

- **channelId** — 专辑id
- **callback** — {@link QTCallback}结果回调,返回类容参考{@link ChannelPrmission}

```
public static void requestRadioList(int pageIndex, QTCallback<List>  
callback)
```

获取广播电台列表

- **Parameters:**

- **pageIndex** — 分页信息从1开始, 默认第一页, 每页30条
- **callback** — {@link QTCallback}结果回调,返回类容参考{@link Radio}

```
public static void requestChannelOnDemandProgramList(int channelId,  
Integer programId,Integer pageIndex, String  
order,QTCallback<QTLISTEntity> callback)
```

获取点播专辑的节目列表

- **Parameters:**

- **channelId** — 专辑id
- **programId** 节目id(带上此参数会给出该节目所在的那一页) 传null则表示没有节目信息
- **pageIndex** 分页信息从1开始, 默认第一页, 每页30条
- **order** 排序 asc(正序默认)/desc(倒序)
- **callback** — {@link QTCallback}结果回调,返回类容参考{@link ChannelProgram}

```
public static void requestChannelAttributes(int categoryId,  
QTCallback<List> callback)
```

获取点播专辑分类下的属性集合

- **Parameters:**

- **categoryId** - 专辑分类id
- **callback** - {@link QTCallback}结果回调,返回类容参考{@link ChannelAttributes}

```
public static void requestRadioDetails(int radioid, QTCallback callback)
```

获取广播电台

- **Parameters:**
 - **radioId** — 电台id
 - **callback** — {@link QTCallback}结果回调,返回类容参考{@link Radio}

```
public static void requestRadioProgramList(int radioid, QTCallback callback)
```

获取广播电台的节目单

- **Parameters:**
 - **radioId** — 电台id
 - **callback** — {@link QTCallback}结果回调,返回类容参考{@link RadioProgramList}

```
public static void requestUserInfo(QTCallback callback)
```

获取用户信息

- **Parameters:** **callback** — {@link QTCallback}结果回调,返回类容参考{@link UserInfo}

```
public static void requestFavChannel(QTCallback callback)
```

获取用户收藏的专辑

- **Parameters:** **callback** — {@link QTCallback}结果回调,返回类容参考{@link FavChannel}

```
public static void addFavChannel(int channelId, QTCallback callback)
```

收藏专辑

- **Parameters:** **callback** — {@link QTCallback}结果回调

```
public static void deleteFavChannel(int channelId, QTCallback callback)
```

取消收藏

- **Parameters:** **callback** — {@link QTCallback}结果回调

```
public static void requestPlayHistory(QTCallback<List> callback)
```

获取用户收听历史

- **Parameters:** **callback** — {@link QTCallback}结果回调,返回类容参考{@link PlayHistory}

```
public static void addPlayRecord(int ts, int channelId, int programId, long playDuration, long position, QTCallback callback)
```

上传收听历史

- **Parameters:** **ts** — 播放停止的时间, 使用Unix时间戳, e.g. 1393257000
- **Parameters:** **channelId** — 专辑id

- **Parameters:** `programId` — 节目id
- **Parameters:** `playDuration` — 本次播放时长, 单位秒
- **Parameters:** `position` — 播放停止的位置 (距节目开头的秒数)
- **Parameters:** `callback` — {@link QTCallback}结果回调

```
public static void search(@NonNull String keyword,@NonNull String type,
String categoryId,@NonNull Integer page,@NonNull final
QTCallback<QTLListEntity> callback)
```

搜索

- **Parameters:** `keyword` — 关键词
- **Parameters:** `type` — 类型: 查询内容的类型, `channel_live`(电台)、`channel_ondemand`(专辑)、`program_ondemand`(点播节目)
- **Parameters:** `categoryId` — 限定专辑分类, 不必传
- **Parameters:** `page` — 分页信息 从1开始
- **Parameters:** `callback` — 回调, 返回类容参考{@link SimpleChannel}

```
public static void requestChannelliveCategories(QTCallback callback)
```

批量获取点播专辑

- **Parameters:** `callback` — {@link QTCallback}结果回调,返回类容参考{@link RadioCategoryList}

```
public static void requestChannelOnDemandList(List channelIds,
QTCallback<HashMap<String, Channel>> callback)
```

获取电台的属性集合

- **Parameters:** `channelIds` — the list of channel ids
- **Parameters:** `callback` — {@link QTCallback}结果回调,返回类容参考{@link Channel}