播放器 SDK

Android 端集成

产品文档

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

Android 端集成
超级播放器

最近更新时间：2022-05-31 15:38:36

产品概述

腾讯云视立方 Android 超级播放器是腾讯云开源的一款播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美市面上各种流行视频 App 的播放软件。

若超级播放器组件满足不了您的业务的个性化需求，且您具有一定的开发经验，可以集成 视立方播放器 SDK，自定义开发播放器界面和播放功能。

准备工作

1. 为了您体验到更完整全面的播放器功能，建议您开通 云点播 相关服务，未注册用户可注册账号 试用。若您不使用云点播服务，可略过此步骤，但集成后仅可使用播放器基础能力。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

通过本文您可以学会

1. 如何集成腾讯云视立方 Android 超级播放器
2. 如何创建和使用播放器

集成准备

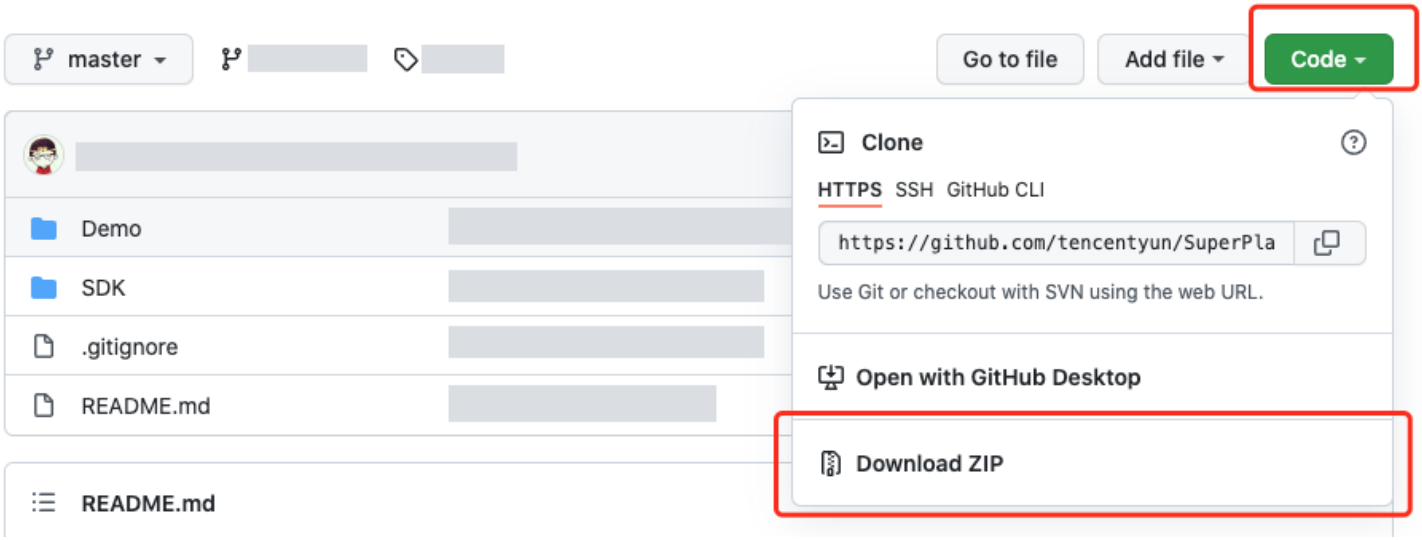
步骤1：项目下载

腾讯云视立方 Android 超级播放器的项目地址是 SuperPlayer_Android。

您可通过 **下载播放器组件 ZIP 包** 或 **Git 命令下载** 的方式下载腾讯云视立方 Android 超级播放器项目工程。

下载播放器组件 ZIP 包

您可以直接下面播放器组件 ZIP包，单击页面的 **Code** > **Download ZIP** 下载。



Git 命令下载

1. 首先确认您的电脑上安装了 Git，如果没有安装，可以参见 Git 安装教程 进行安装。
2. 执行下面的命令把超级播放器组件工程代码 clone 到本地。

```
git clone git@github.com:tencentyun/SuperPlayer_Android.git
```

提示下面的信息表示成功 clone 工程代码到本地。

```
正克隆到 'SuperPlayer_Android'...
remote: Enumerating objects: 2637, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (333/333), done.
remote: Total 2637 (delta 227), reused 524 (delta 170), pack-reused 1993
接收对象中: 100% (2637/2637), 571.20 MiB | 3.94 MiB/s, 完成.
处理 delta 中: 100% (1019/1019), 完成.
```

下载工程后，源码解压后的目录如下：

文件名	作用
LiteAVDemo(Player)	超级播放器 Demo 工程，导入到 Android Studio 后可以直接运行
app	主界面入口
superplayerkit	超级播放器组件（SuperPlayerView），具备播放、暂停、手势控制等常见功能
superplayerdemo	超级播放器 Demo 代码

文件名	作用
common	工具类模块
SDK	视立方播放器 SDK，包括：LiteAVSDK_Player_x.x.x.aar，aar 格式提供的 SDK；LiteAVSDK_Player_x.x.x.zip，lib 和 jar 格式提供的 SDK
Player说明文档(Android).pdf	超级播放器使用文档

步骤2：集成指引

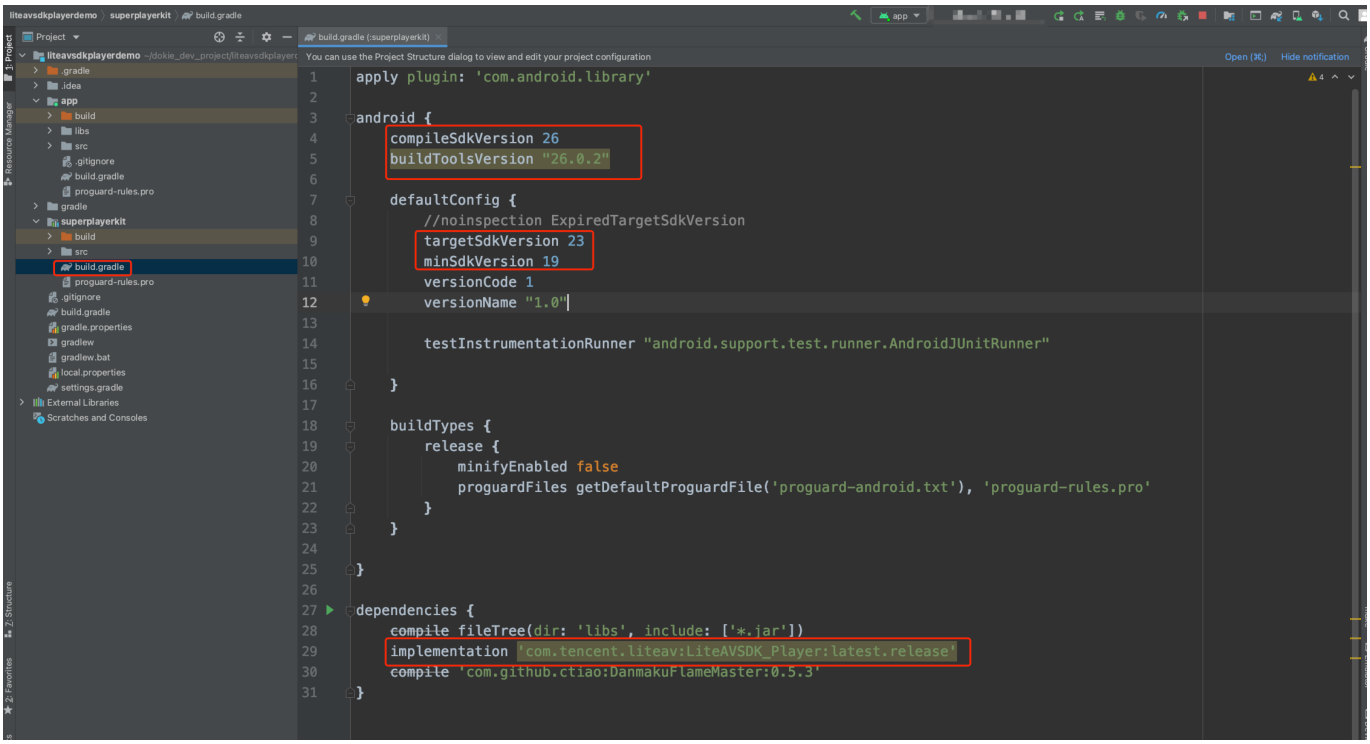
本步骤可指导您如何集成播放器，您可选择使用 Gradle 自动加载的方式，手动下载 aar 再将其导入到您当前的工程或导入 jar 和 so 库的方式集成项目。

Gradle 自动加载（AAR）

1. 下载 SDK + Demo 开发包，项目地址为 Android。
2. 把Demo/superplayerkit这个 module 复制到工程中，然后进行下面的配置：
   - 在工程目录下的setting.gradle 导入superplayerkit。

```
include ':superplayerkit'
```

   - 打开superplayerkit 工程的 build.gradle 文件修改 compileSdkVersion，buildToolsVersion，minSdkVersion，targetSdkVersion 和 rootProject.ext.liteavSdk 的常量值。



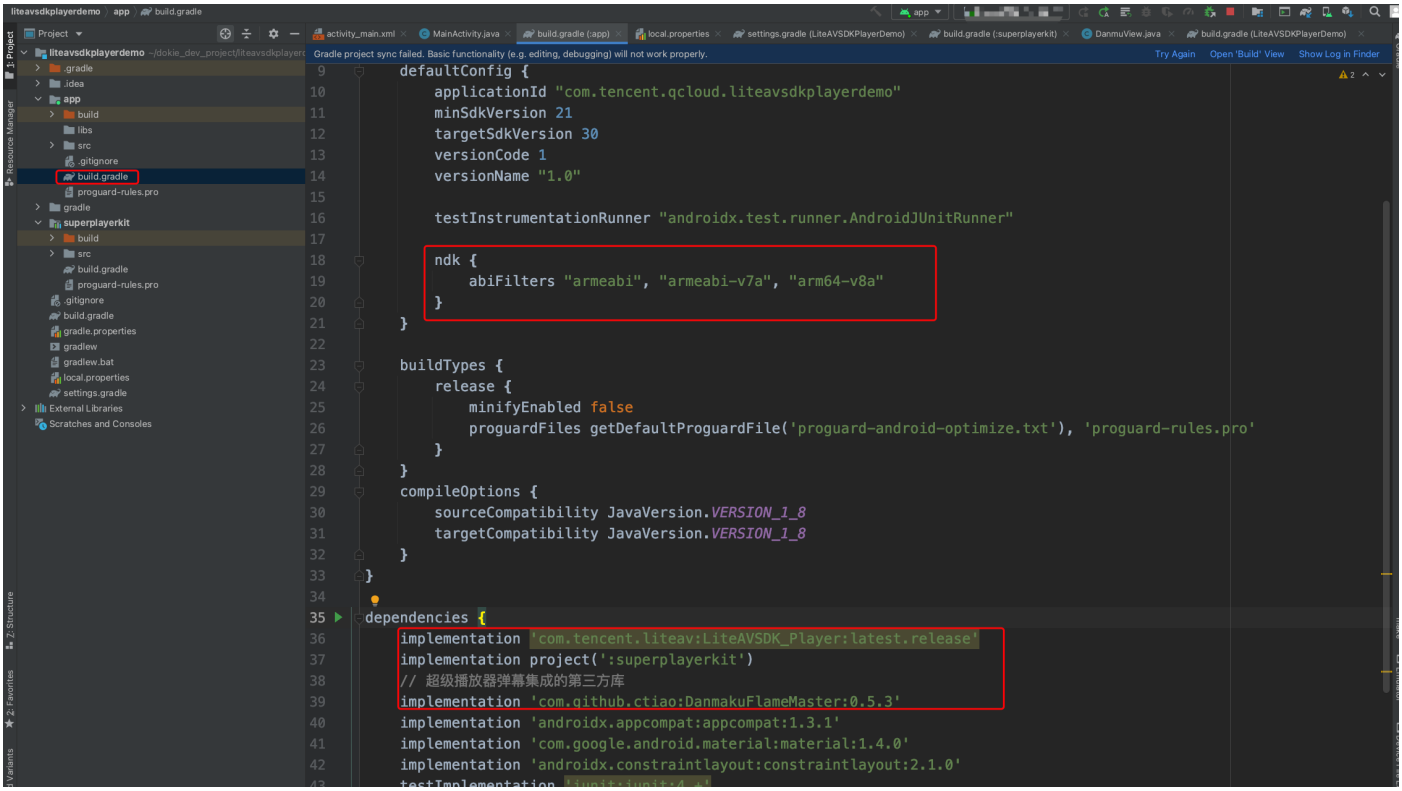

```
compileSdkVersion 26
buildToolsVersion "26.0.2"
```

```
defaultConfig {
targetSdkVersion 23
minSdkVersion 19
}

dependencies {
//如果要集成历史版本，可将 latest.release 修改为对应的版本，例如：8.5.290009
implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

请参见上面的步骤，把common模块导入到项目，并进行配置。

3. 通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK，打开app/build.gradle，进行下面的配置：



i. 在 dependencies 中添加 LiteAVSDK_Player 的依赖。

```
dependencies {
implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
implementation project(':superplayerkit')
// 超级播放器弹幕集成的第三方库
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
}
```

如果您需要集成历史版本的 LiteAVSDK_Player SDK ，可以在 MavenCentral 查看历史版本，然后通过下面的方式进行集成：

```
dependencies {
// 集成8.5.10033 版本LiteAVSDK_Player SDK
implementation 'com.tencent.liteav:LiteAVSDK_Player:8.5.10033'
}
```

ii. 在 app/build.gradle defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、armeabi-v7a 和 arm64-v8a，可根据项目需求配置）。

```
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

如果之前没有使用过9.4以及更早版本的 SDK 的 下载缓存功能（TXVodDownloadManager 中的相关接口），并且不需要在9.5及后续 SDK 版本播放9.4及之前缓存的下载文件，可以不需要该功能的 so 文件，达到减少安装包的体积，例如：在9.4及之前版本使用了 TXVodDownloadManager 类的 setDownloadPath 和 startDownloadUrl 函数下载了相应的缓存文件，并且应用内存储了 TXVodDownloadManager 回调的 getPlayPath 路径用于后续播放，这时候需要 libijkhlscache-master.so 播放该 getPlayPath 路径文件，否则不需要。可以在 app/build.gradle 中添加：

```
packagingOptions {
exclude "lib/armeabi/libijkhlscache-master.so"
exclude "lib/armeabi-v7a/libijkhlscache-master.so"
exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

iii. 在工程目录的 build.gradle 添加 mavenCentral 库。

```
repositories {
mavenCentral()
}
```

4. 单击  Sync Now 按钮同步 SDK，如果您的网络连接 mavenCentral 没有问题，很快 SDK 就会自动下载集成到工程里。

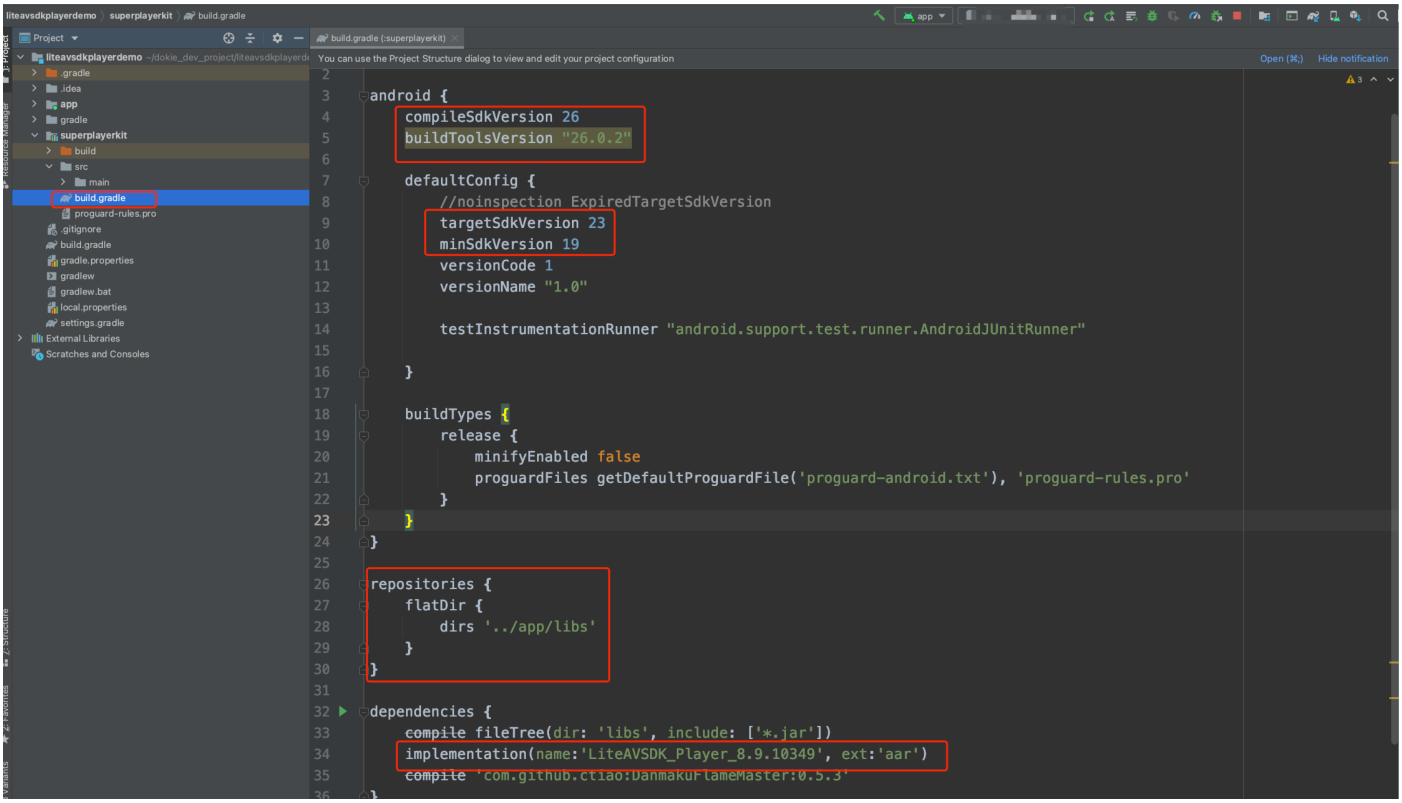
Gradle 手动下载（AAR）

1. 下载 SDK + Demo 开发包，项目地址为 Android。

2. 导入 SDK/LiteAVSDK_Player_XXX.aar（其中 XXX 为版本号）到 app 下面的 libs 文件夹以及复制 Demo/superplayerkit 这个 module 到工程中。
3. 在工程目录下的 setting.gradle 导入superplayerkit。

```
include ':superplayerkit'
```

4. 打开superplayerkit工程的 build.gradle 文件修改 compileSdkVersion，buildToolsVersion，minSdkVersion，targetSdkVersion 和 rootProject.ext.liteavSdk 的常量值。




```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
targetSdkVersion 23
minSdkVersion 19
}

dependencies {
implementation(name:'LiteAVSDK_Player_8.9.10349', ext:'aar')
}
```

请参见上面的步骤，把common模块导入到项目，并进行配置。

- 配置 repositories

```
repositories {
flatDir {
dirs '../app/libs'
}
}
```

5. 在app/build.gradle中添加依赖：

```
compile(name:'LiteAVSDK_Player_8.9.10349', ext:'aar')
implementation project(':superplayerkit')
// 超级播放器弹幕集成的第三方库
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
```

6. 在项目build.gradle中添加：

```
allprojects {
repositories {
flatDir {
dirs 'libs'
}
}
}
```

7. 在 app/build.gradle defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、armeabi-v7a 和 arm64-v8a）。

```
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

如果之前没有使用过9.4以及更早版本的 SDK 的 下载缓存功能（TXVodDownloadManager 中的相关接口），并且不需要在9.5及后续 SDK 版本播放9.4及之前缓存的下载文件，可以不需要该功能的 so 文件，达到减少安装包的体积，例如：在9.4及之前版本使用了 TXVodDownloadManager 类的 setDownloadPath 和 startDownloadUrl 函数下载了相应的缓存文件，并且应用内存储了 TXVodDownloadManager 回调的 getPlayPath 路径用于后续播放，这时候需要 libijkhlscache-master.so 播放该 getPlayPath 路径文件，否则不需要。可以在 app/build.gradle 中添加：

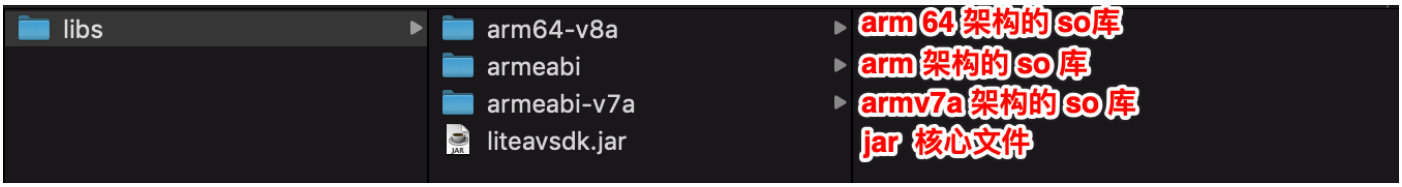
```
packagingOptions {
exclude "lib/armeabi/libijkhlscache-master.so"
exclude "lib/armeabi-v7a/libijkhlscache-master.so"
exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

8. 单击 Sync Now 按钮同步 SDK，完成超级播放器的集成工作。

集成 SDK（jar+so）

如果您不想集成 aar 库，也可以通过导入 jar 和 so 库的方式集成 LiteAVSDK：

1. 下载 SDK + Demo 开发包，项目地址为 Android，下载完成后进行解压。在 SDK 目录找到 SDK/LiteAVSDK_Player_XXX.zip（其中 XXX 为版本号），解压得到 libs 目录，里面包含 jar 文件和 so 文件夹，文件清单如下：

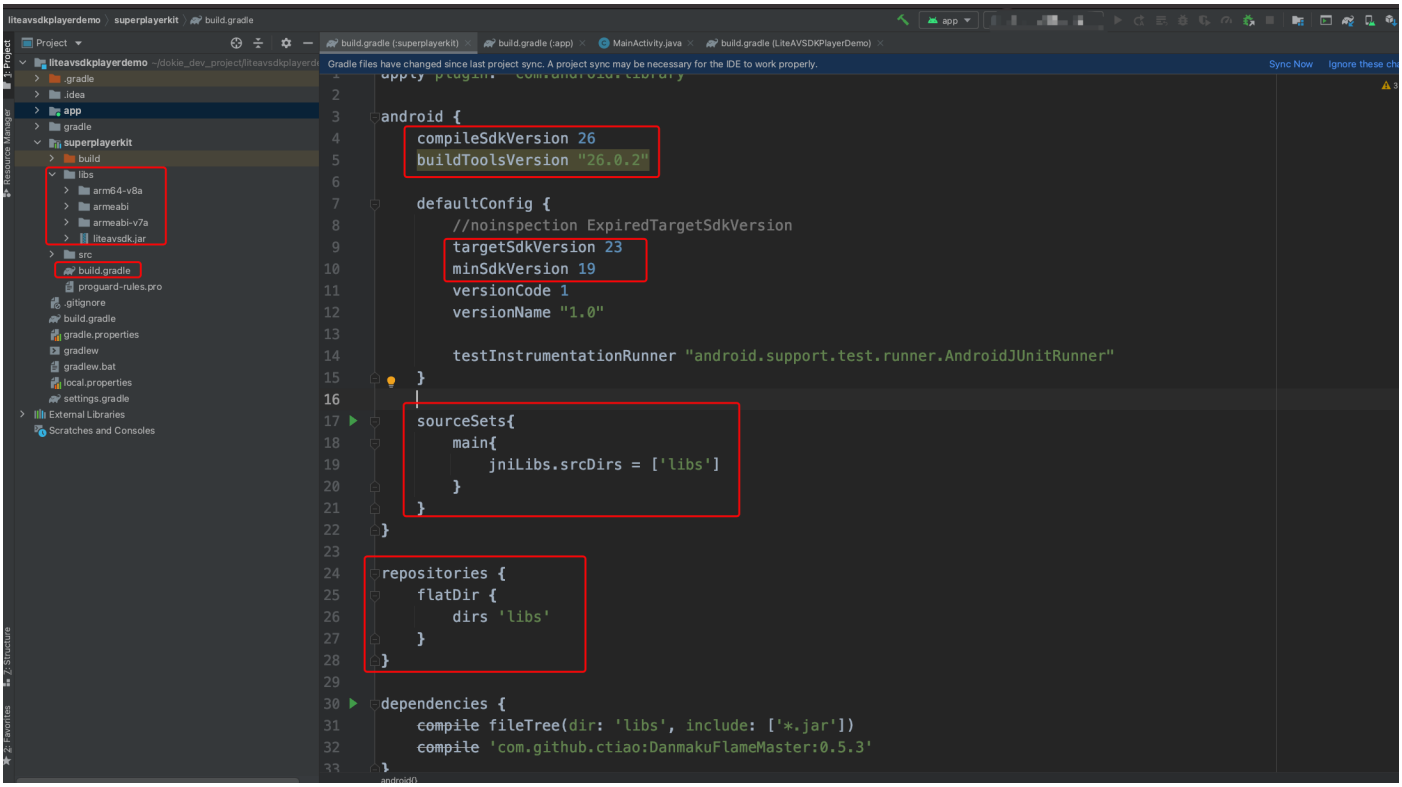


2. 把Demo/superplayerkit这个 module 复制到工程中，然后在工程目录下的 setting.gradle 导入 superplayerkit。

```
include ':superplayerkit'
```

3. 把 步骤1 解压得到的 libs 文件夹复制superplayerkit工程根目录。

4. 修改superplayerkit/build.gradle文件：




```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
targetSdkVersion 23
minSdkVersion 19
}
```

请参见上面的步骤，把common模块导入到项目，并进行配置。

- 配置 sourceSets，添加 so 库引用代码。

```
sourceSets{
main{
jniLibs.srcDirs = ['libs']
}
}
```

- 配置 repositories，添加 flatDir，指定本地仓库路径。

```
repositories {
flatDir {
```

```
dirs 'libs'
}
}
```

5. 在app/build.gradle defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、armeabi-v7a 和 arm64-v8a）。

```
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

6. 单击 Sync Now 按钮同步 SDK，完成 超级播放器的集成工作。

您已经完成了腾讯云视立方 Android 超级播放器项目集成的步骤。

步骤3：配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，LiteAVSDK 需要以下权限：

```
<!--网络权限-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--点播播放器悬浮窗权限-->
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

步骤4：设置混淆规则

在 proguard-rules.pro 文件，将 TRTC SDK 相关类加入不混淆名单：

```
-keep class com.tencent.** { *; }
```

您已经完成了腾讯云视立方 Android 超级播放器 app 权限配置的步骤。

步骤5：使用播放器功能

本步骤，用于指导用户创建和使用播放器，并使用播放器进行视频播放。

1. 创建播放器

播放器主类为 SuperPlayerView ，创建后即可播放视频， 支持集成 FileID 或者 URL 进行播放。在布局文件创建 SuperPlayerView：

```
<!-- 超级播放器-->
<com.tencent.liteav.demo.superplayer.SuperPlayerView
android:id="@+id/superVodPlayerView"
android:layout_width="match_parent"
android:layout_height="200dp" />
```

2. 配置 License 授权

若您已获得相关 License 授权，需在 腾讯云视立方控制台 获取 License URL 和 License Key：

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 Application 类中进行如下设置：

```
public class MApplication extends Application {

@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
TXLiveBase.setListener(new TXLiveBaseListener() {
@Override
public void onLicenceLoaded(int result, String reason) {
Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
```

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

3. 播放视频

本步骤用于指导用户播放视频。腾讯云视立方 Android 超级播放器可用于直播和点播两种播放场景，具体如下：

- 点播播放：超级播放器支持两种点播播放方式，可以 通过 FileID 播放 腾讯云点播媒体资源，也可以直接使用 URL 播放 地址进行播放。
- 直播播放：超级播放器可使用 URL 播放 的方式实现直播播放。通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播URL生成方式可参见 自主拼装直播 URL。**通过 URL 播放（直播、点播）**

URL可以是点播文件播放地址，也可以是直播拉流地址，传入相应 URL 即可播放相应视频文件。

```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073; // 配置 AppId
model.url = "http://your_video_url.mp4"; // 配置您的播放视频url
mSuperPlayerView.playWithModel(model);
```

通过 FileID 播放（点播）

视频 FileId 在一般是在视频上传后，由服务器返回：

a. 客户端视频发布后，服务器会返回 FileId 到客户端。

b. 服务端视频上传时，在 确认上传 的通知中包含对应的 FileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件，查看 FileId。如下图所示，ID 即表示 FileId：

> ⚠ 注意：
>
> - 通过 FileID 播放时，需要首先使用 Adaptive-HLS(10) 转码模板对视频进行转码，或者使用超级播放器签名 psign 指定播放的视频，否则可能导致视频播放失败。转码教程和说明可参见 用超级播放器播放视频，psign 生成教程可参见 psign 教程。

- 若您在通过 FileID 播放时出现“no v4 play info”异常，则说明您可能存在上述问题，建议您根据上述教程调整。同时您也可以直接获取源视频播放链接，通过 URL 播放 的方式实现播放。
- **未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放。**

```
//在未开启防盗链进行播放的过程中，如果出现了“no v4 play info”异常，建议您使用Adaptive-HLS(10)转码模板对视频进行转码，或直接获取源视频播放链接通过url方式进行播放。

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// 配置 AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // 配置 FileId
//私有加密播放需填写 psign， psign 即超级播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
//model.videoId.pSign = @"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
[_playerView playWithModel:model];
```

4. **退出播放**

当不需要播放器时，调用 resetPlayer 清理播放器内部状态，释放内存。

```
mSuperPlayerView.resetPlayer();
```

至此，您已经完成了腾讯云视立方 Android 超级播放器创建、播放视频和退出播放的能力即成。

功能使用

本章将为您介绍几种常见的播放器功能使用方式，更为完整的功能使用方式可参见 Demo 体验，超级播放器支持的功能可参见 能力清单。

1、全屏播放

超级播放器支持全屏播放，在全屏播放场景内，同时支持锁屏、手势控制音量和亮度、弹幕、截屏、清晰度切换等功能设置。功能效果可在 **腾讯云视立方 App > 播放器 > 超级播放器** 中体验，单击界面右下角即可进入全屏播放界面。

在窗口播放模式下，可通过调用下述接口进入全屏播放模式：

```
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.FULLSCREEN);
```

全屏播放界面功能介绍

返回窗口

单击返回，即可返回至窗口播放模式。

```
//单击后触发下面的接口
mControllerCallback.onBackPressed(SuperPlayerDef.PlayerMode.FULLSCREEN);
onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

锁屏

锁屏操作可以让用户进入沉浸式播放状态。

```
//单击后触发的接口
toggleLockState();
```

弹幕

打开弹幕功能后屏幕上会有用户发送的文字飘过。

```
// 步骤一：向弹幕View中添加一条弹幕
addDanmaku(String content, boolean withBorder);
// 步骤二：打开或者关闭弹幕
toggleBarrage();
```

截屏

超级播放器提供播放过程中截取当前视频帧功能，您可以把图片保存起来进行分享。单击图片4处按钮可以截屏，您可以在 mSuperPlayer.snapshot 接口进行保存截取的图片。

```
mSuperPlayer.snapshot(new TXLivePlayer.ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bitmap) {
//在这里可以保存截图
}
});
```

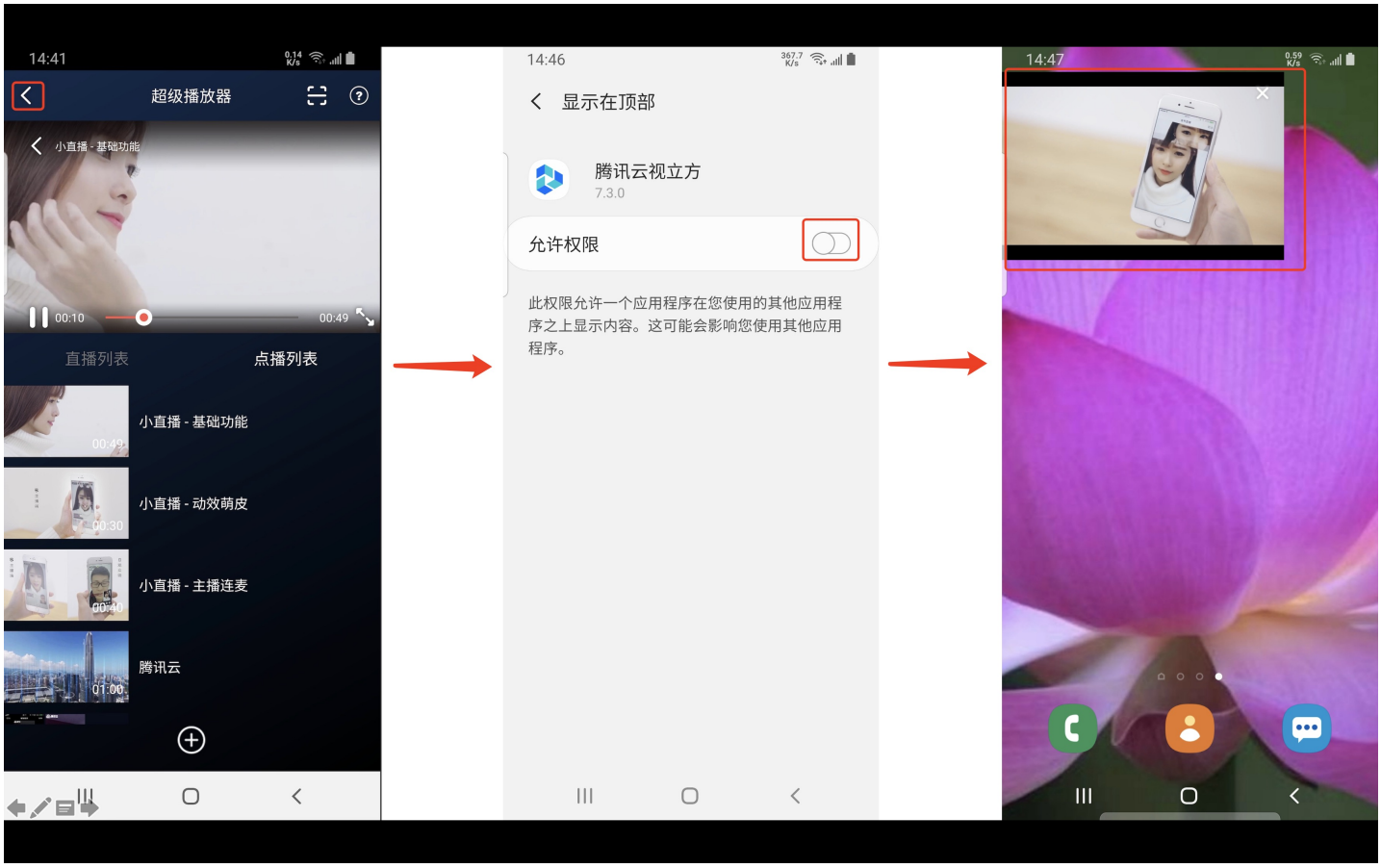
清晰度切换

用户可以根据需求选择不同的视频播放清晰度，如高清、标清或超清等。

```
//单击后触发的显示清晰度view代码接口
showQualityView();
//单击清晰度选项的回调接口为
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
// 清晰度ListView的单击事件
VideoQuality quality = mList.get(position);
mCallback.onQualitySelect(quality);
}
});
//最终改变清晰度的回调
@Override
public void onQualityChange(VideoQuality quality) {
mFullScreenPlayer.updateVideoQuality(quality);
mSuperPlayer.switchStream(quality);
}
```

2、悬浮窗播放

超级播放器支持悬浮窗小窗口播放，可在切换到其它应用时，不打断视频播放功能。功能效果可在 **腾讯云视立方 App** > **播放器** > **超级播放器** 中体验，单击界面左上角**返回**，即可体验悬浮窗播放功能。



悬浮窗播放依赖于 AndroidManifest 中的以下权限：

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

```
// 切换悬浮窗触发的代码接口
mSuperPlayerView.switchPlayMode(SuperPlayerDef.PlayerMode.FLOAT);
//单击浮窗返回窗口触发的代码接口
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

3、视频封面

超级播放器支持用户自定义视频封面，用于在视频接收到首帧画面播放回调前展示。功能效果可在 **腾讯云视立方 App** > **播放器** > **超级播放器** > **自定义封面演示** 视频中体验。

- 当超级播放器设置为自动播放模式PLAY_ACTION_AUTO_PLAY时，视频自动播放，此时将在视频首帧加载出来之前展示封面；
- 当超级播放器设置为手动播放模式PLAY_ACTION_MANUAL_PLAY时，需用户单击**播放**后视频才开始播放。在单击**播放**前将展示封面；在单击**播放**后到视频首帧加载出来前也将展示封面。

视频封面支持使用网络 URL 地址或本地 File 地址，使用方式可参见下述指引。若您通过 FileID 的方式播放视频，则可直接在云点播内配置视频封面。

```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = "您的appid";
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "您的fileId";
//播放模式，可设置自动播放模式：PLAY_ACTION_AUTO_PLAY，手动播放模式：PLAY_ACTION_MANUAL_PLA
Y
```

```
model.playAction = PLAY_ACTION_MANUAL_PLAY;
//设定封面的地址为网络url地址，如果coverPictureUrl不设定，那么就会自动使用云点播控制台设置的封面
model.coverPictureUrl = "http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500005830/cc1e28208602268011087336518/MXUW1a5I9TsA.png"
mSuperPlayerView.playWithModel(model);
```

4、视频列表轮播

超级播放器支持视频列表轮播，即在给定一个视频列表后：

- **支持按顺序循环播放列表中的视频，播放过程中支持自动播放下一集也支持手动切换到下一个视频。**
- **列表中最后一个视频播放完成后将自动开始播放列表中的第一个视频。**

功能效果可在 腾讯云视立方 App> 播放器 > 超级播放器 > 视频列表轮播演示视频中体验。


```
//步骤1:构建轮播的List<SuperPlayerModel>
ArrayList<SuperPlayerModel> list = new ArrayList<>();
SuperPlayerModel model = new VideoModel();
model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071568";
```

```
list.add(model);

model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071679";
list.add(model);
//步骤2：调用轮播接口
mSuperPlayerView.playWithModelList(list, true, 0);
```

```
public void playWithModelList(List<SuperPlayerModel> models, boolean isLoopPlayList, int index);
```

接口参数说明

参数名	类型	描述
models	List	轮播数据列表
isLoopPlayList	boolean	是否循环
index	int	开始播放的 SuperPlayerModel 索引

5、视频试看

超级播放器支持视频试看功能，可以适用于非 VIP 试看等场景，开发者可以传入不同的参数来控制视频试看时长、提示信息、试看结束界面等。功能效果可在 **腾讯云视立方 App> 播放器 > 超级播放器 > 试看功能演示** 视频中体验。

方法一：

```
//步骤1：创建视频mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss，开通 VIP 观看完整视频",15);
mode.vipWatchMode = vipWatchModel;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModel(mode);
```

方法二：

```
//步骤1：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss，开通 VIP 观看完整视频",15);
//步骤2：调用设置试看功能方法
mSuperPlayerView.setVipWatchModel(vipWatchModel);
```

```
public VipWatchModel(String tipStr, long canWatchTime)
```

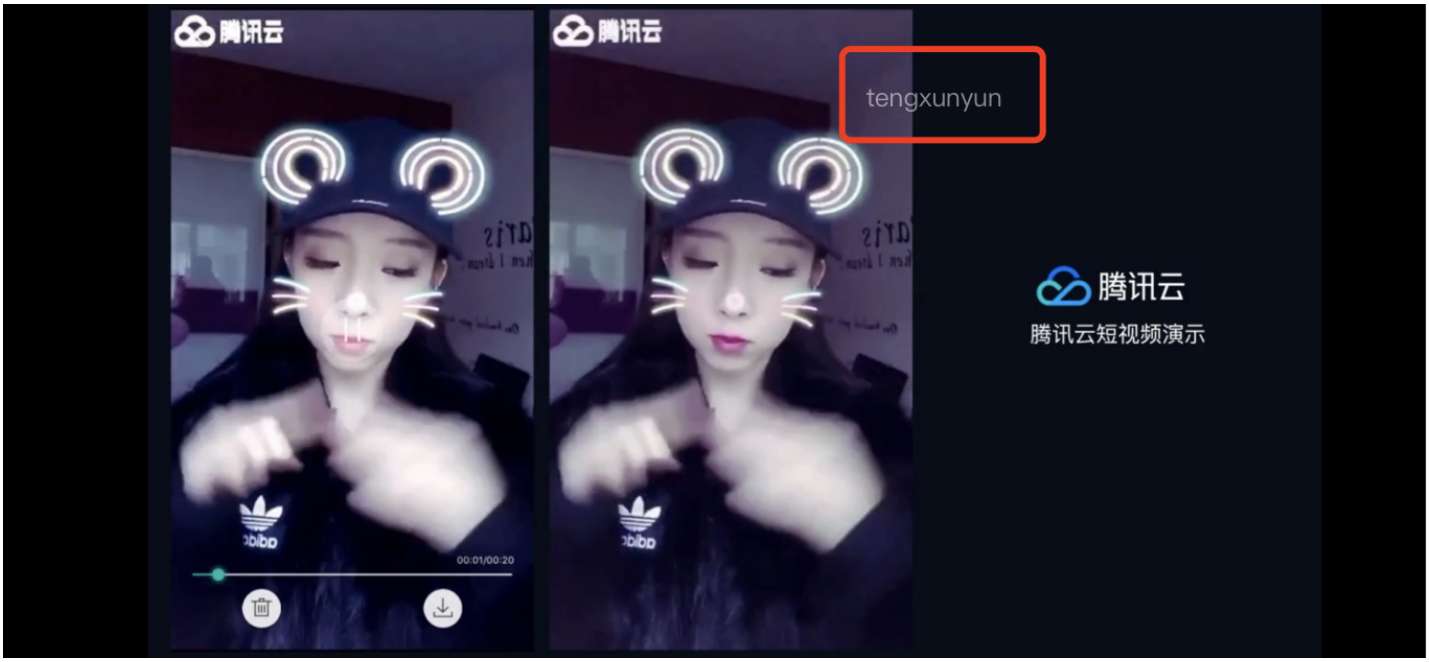
VipWatchModel 接口参数说明：

参数名	类型	描述
tipStr	String	试看提示信息

参数名	类型	描述
canWatchTime	Long	试看时长，单位为妙

6、动态水印

超级播放器支持在播放界面添加不规则跑动的文字水印，有效防盗录。全屏播放模式和窗口播放模式均可展示水印，开发者可修改水印文本、文字大小、颜色。功能效果可在 **腾讯云视立方 App** > **播放器** > **超级播放器** > **动态水印** 演示视频中体验。



方法一：

```
//步骤1：创建视频mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建水印信息mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, Color.parse
Color("#80FFFFFF"));
mode.dynamicWaterConfig = dynamicWaterConfig;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModel(mode);
```

方法二：

```
//步骤1：创建水印信息mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, Color.parse
Color("#80FFFFFF"));
```

```
//步骤2：调用设置动态水印功能方法
mSuperPlayerView.setDynamicWatermarkConfig(dynamicWaterConfig);
```

```
public DynamicWaterConfig(String dynamicWatermarkTip, int tipTextSize, int tipTextColor)
```

接口参数说明

参数名	类型	描述
dynamicWatermarkTip	String	水印文本信息
tipTextSize	int	文字大小
tipTextColor	int	文字颜色

Demo 体验

更多完整功能可直接运行工程 Demo，或扫码下载移动端 Demo 腾讯云视立方 App 体验。

运行工程 Demo

1. 在 Android Studio 的导航栏选择 **File** > **Open**，在弹框中选择 **Demo** 工程目录：
   $SuperPlayer_Android/Demo，待成功导入 Demo 工程后，单击 **Run app**，即可成功运行 Demo 。
2. 成功运行 Demo 后如下图，进入**播放器** > **超级播放器**，可体验播放器功能。

腾讯云视立方 App

在 **腾讯云视立方 App** > **播放器** 中可体验更多超级播放器功能。

超级播放器 Adapter

最近更新时间：2022-04-14 10:57:52

腾讯云视立方 Android 超级播放器 Adapter 为云点播提供给客户希望使用第三方播放器或自研播放器开发的对接云 PaaS 资源的播放器插件，常用于有自定义播放器功能需求的用户。

SDK 下载

腾讯云视立方 Andriod 超级播放器 Adapter SDK 和 Demo 项目下载地址 TXCPlayerAdapterSDK_Android。

集成指引

SDK 集成

集成 SDK，拷贝 TXCPlayerAdapter-release-1.0.0.aar 到 libs 目录，添加依赖项：

```
implementation(name:'TXCPlayerAdapter-release-1.0.0', ext:'aar')
```

添加混淆脚本：

```
-keep class com.tencent.** { *; }
```

使用播放器

变量声明，播放器主类为 ITXCPlayerAssistor ，创建后即可播放视频。

fileId 一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 fileId 到客户端。
2. 服务端视频上传，在 确认上传 的通知中包含对应的 fileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件。点开后在右侧视频详情中，可以看到相关参数。

```
//psign 即超级播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
private String mFileId, mPSign;
ITXCPlayerAssistor mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId, mPSign);
```

初始化：

```
// 初始化
TXCPlayerAdapter.init(appId); //appid 在腾讯云点播申请
TXCPlayerAdapter.setLogEnable(true); //开启log

mSuperPlayerView = findViewById(R.id.sv_videoplayer);
mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId, mPSign);
```

请求视频信息和播放：

```
mPlayerAssistor.requestVideoInfo(new ITXCRequestVideoInfoCallback() {

@Override
public void onError(int errCode, String msg) {
Log.d(TAG, "onError msg = " + msg);
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(VideoActivity.this, "onError msg = " + msg, Toast.LENGTH_SHORT).show();
}
});
}

@Override
public void onSuccess() {
Log.d(TAG, "onSuccess");
TXCStreamingInfo streamingInfo = mPlayerAssistor.getStreamingInfo();
Log.d(TAG, "streamingInfo = " + streamingInfo);
runOnUiThread(new Runnable() {
@Override
public void run() {
if (mPlayerAssistor.getStreamingInfo() != null) {
//播放视频
mSuperPlayerView.play(mPlayerAssistor.getStreamingInfo().playUrl);
} else {
Toast.makeText(VideoActivity.this, "streamInfo = null", Toast.LENGTH_SHORT).show();
}
}
});
```

```
}
});
```

使用完后销毁 Player。

```
TXCPlayerAdapter.destroy();
```

SDK 接口说明

初始化 TXCPlayerAdatper

初始化 Adapter（每次）。

接口

```
TXCPlayerAdapter.init(String appId);
```

参数说明

appId：填写 appid（如果使用了子应用，则填 subappid）。

销毁 TXCPlayerAdatper

销毁 Adapter，当程序退出后调用。

接口

```
TXCPlayerAdapter.destroy();
```

创建播放器辅助类

通过播放器辅助类可以获取播放 fileId 相关信息以及处理 DRM 加密接口等。

接口

```
ITXCPlayerAssistor playerAssistor = TXCPlayerAdapter.createPlayerAssistor(String fileId, String pSig
n);
```

参数说明

参数名	类型	描述

参数名	类型	描述
fileId	String	要播放的视频 fileId
pSign	String	超级播放器签名

销毁播放器辅助类

销毁辅助类，在退出播放器或者切换了下一个视频播放的时候调用。

接口

```
TXCPlayerAdapter.destroyPlayerAssistor(ITXCPlayerAssistor assistor);
```

请求视频播放信息

本接口会请求腾讯云点播服务器，获取播放视频的流信息等。

接口

```
playerAssistor.requestVideoInfo(ITXCRequestVideoInfoCallback callback);
```

参数说明

参数名	类型	描述
callback	ITXCRequestVideoInfoCallback	异步回调函数

获取视频的基本信息

获取视频信息， 必须是在 playerAssistor.requestPlayInfo 回调之后才生效。

接口

```
TXCVideoBasicInfo playerAssistor.getVideoBasicInfo();
```

参数说明

TXCVideoBasicInfo 参数如下：

参数名	类型	描述
name	String	视频名称

参数名	类型	描述
duration	Float	视频时长，单位：秒
description	String	视频描述
coverUrl	String	视频封面

获取视频流信息

获取视频流信息列表，必须是在 playerAssistor.requestPlayInfo 回调之后才生效。

接口

```
TXCStreamingInfo playerAssistor.getStreamimgInfo();
```

参数说明

TXCStreamingInfo

参数名	类型	描述
playUrl	String	播放 URL
subStreams	List	自适应码流子流信息，类型为 SubStreamInfo

SubStreamInfo 参数如下：

参数名	类型	描述
type	String	子流的类型，目前可能的取值仅有 video
width	Int	子流视频的宽，单位：px
height	Int	子流视频的高，单位：px
resolutionName	String	子流视频在播放器中展示的规格名

获取关键帧打点信息

获取视频关键帧打点信息，必须是在 playerAssistor.requestPlayInfo 回调之后才生效。

接口

```
List<TXCKeyFrameDescInfo> playerAssistor.getKeyFrameDescInfo();
```

参数说明

TXCKeyFrameDescInfo 参数如下：

参数名	类型	描述
timeOffset	Float	1.1
content	String	"片头开始..."

获取缩略图信息

获取缩略图信息，必须是在 playerAssistor.requestPlayInfo 回调之后才生效。

接口

```
TXCImageSpriteInfo playerAssistor.getImageSpriteInfo();
```

参数说明

TCXImageSpriteInfo 参数如下：

参数名	类型	描述
imageUrls	List	缩略图下载 URL 数组，类型为 String
webVttUrl	String	缩略图 VTT 文件下载 URL

定制开发
直播场景
接入文档

最近更新时间：2022-05-31 15:38:05

本文档主要介绍使用腾讯云视立方 SDK 实现直播播放的方法。

示例代码

针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

所属平台	GitHub 地址
iOS	Github
Android	Github

准备工作

1. 为了您更好的产品体验，建议您开通 云直播 相关服务。若您不使用云直播服务，可略过此步骤。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

通过本文你可以学会

- 如何集成腾讯云视立方 Android 播放器 SDK
- 如何使用播放器 SDK 进行直播播放
- 如何使用播放器 SDK 底层能力实现更多直播播放功能

SDK集成

步骤1：下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

步骤2：配置 License 授权

若您已获得相关 License 授权，需在 腾讯云视立方控制台 获取 License URL 和 License Key：

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 Application 类中进行如下设置：

```
public class MApplication extends Application {

@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
TXLiveBase.setListener(new TXLiveBaseListener() {
@Override
public void onLicenceLoaded(int result, String reason) {
Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
}
});
}
}
```

步骤3：添加 View

SDK 默认提供 TXCloudVideoView 用于视频渲染，我们第一步要做的就是在布局 xml 文件里加入如下一段代码：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/video_view"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_centerInParent="true"
android:visibility="gone"/>
```

步骤4：创建 Player

视频云 SDK 中的 **TXLivePlayer** 模块负责实现直播播放功能，并使用 **setPlayerView** 接口将它与我们刚添加到界面上的 **video_view** 控件进行关联。

```
//mPlayerView 即步骤1中添加的界面 view
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);

//创建 player 对象
TXLivePlayer mLivePlayer = new TXLivePlayer(getActivity());

//关键 player 对象与界面 view
mLivePlayer.setPlayerView(mView);
```

步骤5：启动播放

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startPlay(flvUrl, TXLivePlayer.PLAY_TYPE_LIVE_FLV); //推荐 FLV
```

可选值	枚举值	含义
PLAY_TYPE_LIVE_RTMP	0	传入的 URL 为 RTMP 直播地址
PLAY_TYPE_LIVE_FLV	1	传入的 URL 为 FLV 直播地址
PLAY_TYPE_LIVE_RTMP_ACC	5	低延迟链路地址（仅适合于连麦场景）
PLAY_TYPE_VOD_HLS	3	传入的 URL 为 HLS（m3u8）播放地址

> 说明：
> 在 App 上我们不推荐使用 HLS 这种播放协议播放直播视频源（虽然它很适合用来做点播），因为延迟太高，在 App 上推荐使用 LIVE_FLV 或者 LIVE_RTMP 播放协议。

步骤6：结束播放

结束播放时**需要销毁 view 控件**，尤其是在下次 startPlay 之前，否则**会产生大量的内存泄露以及闪屏问题**。

同时，在退出播放界面时，**需要调用渲染 View 的onDestroy()函数**，否则**可能会产生内存泄露和 Receiver not registered** 报警。

```
@Override
public void onDestroy() {
super.onDestroy();
mLivePlayer.stopPlay(true); // true 代表清除最后一帧画面
mView.onDestroy();
}
```

stopPlay 的布尔型参数含义为：“是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。

如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

功能使用

本节将介绍常见直播播放功能的使用方式。

1、画面调整

- **view：大小和位置**
  如需修改画面的大小及位置，直接调整**步骤1**中添加的 video_view 控件的大小和位置即可。
- **setRenderMode：铺满或适应**

可选值	含义
RENDER_MODE_FULL_FILL_SCREEN	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全
RENDER_MODE_ADJUST_RESOLUTION	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边

- **setRenderRotation：画面旋转**

可选值	含义
RENDER_ROTATION_PORTRAIT	正常播放（Home 键在画面正下方）
RENDER_ROTATION_LANDSCAPE	画面顺时针旋转270度（Home 键在画面正左方）

```
// 设置填充模式
mLivePlayer.setRenderMode(TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION);
// 设置画面渲染方向
mLivePlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_LANDSCAPE);
```

最长边填充

完全填充

横屏模式

2、暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这跟点播是有很大不同的（点播播放器的暂停和继续与播放本地视频文件时的表现相同）。

```
// 暂停
mLivePlayer.pause();
// 继续
mLivePlayer.resume();
```

3、消息接收

此功能可以在推流端将一些自定义 message 随着音视频线路直接下发到观众端，适用场景如下：

- 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”完美同步。

- 秀场直播：推流端将**歌词**下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
- 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈划线。

通过如下方案可以使用此功能：

- TXLivePlayConfig 中的 **setEnableMessage** 开关置为 **true**。
- TXLivePlayer 通过 **TXLivePlayListener** 监听消息，消息编号：**PLAY_EVT_GET_MESSAGE（2012）**

```
//Android 示例代码
mTXLivePlayer.setPlayListener(new ITXLivePlayListener() {
@Override
public void onPlayEvent(int event, Bundle param) {
if (event == TXLiveConstants.PLAY_ERR_NET_DISCONNECT) {
roomListenerCallback.onDebugLog("[AnswerRoom] 拉流失败：网络断开");
roomListenerCallback.onError(-1, "网络断开，拉流失败");
}
else if (event == TXLiveConstants.PLAY_EVT_GET_MESSAGE) {
String msg = null;
try {
msg = new String(param.getByteArray(TXLiveConstants.EVT_GET_MSG), "UTF-8");
roomListenerCallback.onRecvAnswerMsg(msg);
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}
}
@Override
public void onNetStatus(Bundle status) {
}
});
```

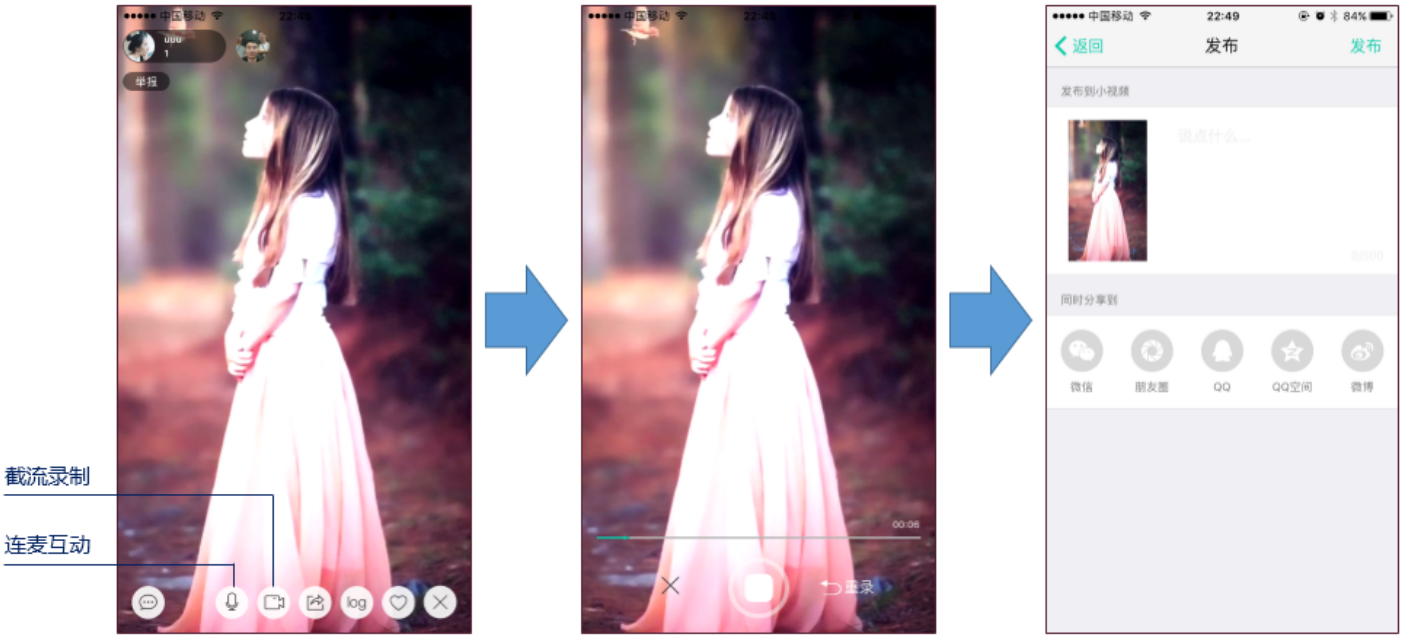
4、屏幕截图

通过调用 **snapshot**，您可以截取当前直播画面为一帧屏幕，此功能只会截取当前直播流的视频画面，如果您需要截取当前的整个 UI 界面，请调用 Android 的系统 API 来实现。


```
mLivePlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
//获取到截图 bitmap
}
}
});
```

5、截流录制

截流录制是直播播放场景下的一种扩展功能：观众在观看直播时，可以通过单击录制按钮把一段直播的内容录制下来，并通过视频分发平台（如腾讯云的点播系统）发布出去，这样就可以在微信朋友圈等社交平台上以 UGC 消息的形式进行传播。




```
//指定一个 ITXVideoRecordListener 用于同步录制的进度和结果
mLivePlayer.setVideoRecordListener(recordListener);
//启动录制，可放于录制按钮的响应函数里，目前只支持录制视频源，弹幕消息等等目前还不支持
mLivePlayer.startRecord(int recordType);
// ...
// ...
//结束录制，可放于结束按钮的响应函数里
mLivePlayer.stopRecord();
```

- 录制的进度以时间为单位，由 ITXVideoRecordListener 的 onRecordProgress 通知出来。
- 录制好的文件以 MP4 文件的形式，由 ITXVideoRecordListener 的 onRecordComplete 通知出来。
- 视频的上传和发布由 TXUGCPublish 负责，具体使用方法可以参考 短视频 – 文件发布。

6、清晰度无缝切换

日常使用中，网络情况在不断发生变化。在网络较差的情况下，最好适度降低画质，以减少卡顿；反之，网速比较好，可以观看更高画质。
传统切流方式一般是重新播放，会导致切换前后画面衔接不上、黑屏、卡顿等问题。使用无缝切换方案，在不中断直播的情况下，能直接切到另条流上。

清晰度切换在直播开始后，任意时间都可以调用。调用方式如下

```
// 正在播放的是流 http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab11e791eae435c87f075e.flv，
```

```
// 现切换到码率为900kbps的新流上
mLivePlayer.switchStream("http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab11e791eae435c87f075e_900.flv");
```

> 说明：
> 清晰度无缝切换功能需要在后台配置 PTS 对齐，如您需要可 提交工单 申请使用。

7、直播回看

时移功能是腾讯云推出的特色能力，可以在直播过程中，随时回退到任意直播历史时间点观看，并能在此时间点一直观看直播。非常适合游戏、球赛等互动性不高，但观看连续性较强的场景。

```
// 设置直播回看前，先调用 startPlay
// 开始播放 ...
TXLiveBase.setAppID("1253131631"); // 配置 appId
mLivePlayer.prepareLiveSeek(); // 后台请求直播起始时间
```

配置正确后，在 PLAY_EVT_PLAY_PROGRESS 事件里，当前进度就不是从0开始，而是根据实际开播时间计算而来。
调用 seek 方法，就能从历史事件点重新直播

```
mLivePlayer.seek(600); // 从第10分钟开始播放
```

接入时移需要在后台打开以下配置：

- 录制：配置时移时长、时移储存时长。
- 播放：时移获取元数据。

时移功能处于公测申请阶段，如您需要可 提交工单 申请使用。

8、延时调节

腾讯云 SDK 的直播播放（LVB）功能，并非基于 ffmpeg 做二次开发， 而是采用了自研的播放引擎，所以相比于开源播放器，在直播的延迟控制方面有更好的表现，我们提供了三种延迟调节模式，分别适用于：秀场、游戏以及混合场景。

- 三种模式的特性对比

控制模式	卡顿率	平均延迟	适用场景	原理简述

控制模式	卡顿率	平均延迟	适用场景	原理简述
极速模式	较流畅偏高	2s - 3s	美女秀场（冲顶大会）	在延迟控制上有优势，适用于对延迟大小比较敏感的场景
流畅模式	卡顿率最低	>= 5s	游戏直播（企鹅电竞）	对于超大码率的游戏直播（例如吃鸡）非常适合，卡顿率最低
自动模式	网络自适应	2s - 8s	混合场景	观众端的网络越好，延迟就越低；观众端网络越差，延迟就越高

- **三种模式的对接代码**

```
TXLivePlayConfig mPlayConfig = new TXLivePlayConfig();
//
//自动模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(5);
//
//极速模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(1);
//
//流畅模式
mPlayConfig.setAutoAdjustCacheTime(false);
mPlayConfig.setCacheTime(5);
//
mLivePlayer.setConfig(mPlayConfig);
//设置完成之后再启动播放
```

> **说明：**
> 更多关于卡顿和延迟优化的技术知识，可以阅读 视频卡顿怎么办？

9、超低延时播放

支持400ms左右的超低延迟播放，是腾讯云直播播放器的一个特点，它可以用于一些对时延要求极为苛刻的场景，例如**远程夹娃娃或者主播连麦**等，关于这个特性，您需要知道：

- **该功能是不需要开通的**
  该功能并不需要提前开通，但是要求直播流必须位于腾讯云，跨云商实现低延时链路的难度不仅仅是技术层面的。
- **播放地址需要带防盗链**
  播放 URL 不能用普通的 CDN URL，必须要带防盗链签名，防盗链签名的计算方法见 txTime 与 txSecret。
- **播放类型需要指定 ACC**
  在调用 startPlay 函数时，需要指定 type 为 **PLAY_TYPE_LIVE_RTMP_ACC**，SDK 会使用 RTMP-UDP 协议拉取直播流。
- **该功能有并发播放限制**
  目前最多同时**10路**并发播放，设置这个限制的原因并非是技术能力限制，而是希望您只考虑在互动场景中使用（例如连麦时只给主播使用， 或者夹娃娃直播中只给操控娃娃机的玩家使用），避免因为盲目追求低延时而产生不必要的费用损失（低延迟线路的价格要贵于 CDN 线路）。
- **Obs 的延时是不达标的**
  推流端如果是 TXLivePusher，请使用 setVideoQuality 将 quality 设置为 MAIN_PUBLISHER 或者 VIDEO_CHAT。Obs 的推流端积压比较严重，是无法达到低延时效果的。
- **该功能按播放时长收费**
  本功能按照播放时长收费，费用跟拉流的路数有关系，跟音视频流的码率无关，具体价格请参考 价格总览。

10、获取视频信息

播放器 SDK 通过 URL 字符串播放视频，URL 中本身不包含视频信息。为获取相关信息，需要通过访问云端服务器加载到相关视频信息，因此 SDK 只能以事件通知的方式将视频信息发送到您的应用程序中，更多内容参见 事件监听。

例如您可以通过 onNetStatus 的 NET_STATUS_VIDEO_WIDTH 和 NET_STATUS_VIDEO_HEIGHT 获取视频的宽和高。具体使用方法见 状态反馈（onNetStatus）。

事件监听

您可以为 TXLivePlayer 对象绑定一个 TXLivePlayListener，之后 SDK 的内部状态信息均会通过 onPlayEvent（事件通知）和 onNetStatus（状态反馈）通知给您。

事件通知（onPlayEvent）

1. 播放事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度 和总体时长

事件 ID	数值	含义说明
PLAY_EVT_PLAY_LOADING	2007	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件
PLAY_EVT_VOD_LOADING_END	2014	视频播放 loading 结束，视频继续播放

不要在收到 PLAY_LOADING 后隐藏播放画面：因为 PLAY_LOADING -> PLAY_BEGIN 的时间长短是不确定的，可能是5s或5ms，有些用户考虑在 LOADING 时隐藏画面， BEGIN 时显示画面，会造成严重的画面闪烁（尤其是直播场景下）。**推荐的做法是在视频播放画面上叠加一个半透明的 loading 动画。**

2. 结束事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败

如何判断直播已结束?

基于各种标准的实现原理不同，很多直播流通常没有结束事件（2006）抛出，此时可预期的表现是：主播结束推流后，SDK 会很快发现数据流拉取失败（WARNING_RECONNECT），然后开始重试，直至三次重试失败后抛出 PLAY_ERR_NET_DISCONNECT 事件。

因此，2006和-2301都要监听，用来作为直播结束的判定事件。

3. 警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件 ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解

状态反馈（onNetStatus）

通知每秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前 SDK 内部的一些具体情况，以便您能对当前网络状况和视频信息等有所了解。

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时 CPU 使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 – 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 – 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_CACHE_SIZE	缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器 IP

API 文档

最近更新时间：2022-01-06 15:37:09

TXLivePlayer

视频播放器

请参见 TXLivePlayer。

主要负责将直播流的音视频画面进行解码和本地渲染，包含如下技术特点：

- 针对腾讯云的拉流地址，可使用低延时拉流，实现直播连麦等相关场景。
- 针对腾讯云的拉流地址，可使用直播时移功能，能够实现直播观看与时移观看的无缝切换。
- 支持自定义的音视频数据处理，让您可以根据项目需要处理直播流中的音视频数据后，进行渲染以及播放。

SDK 基础函数

API	描述
TXLivePlayer	创建 TXLivePlayer 实例。
setConfig	设置 TXLivePlayer 播放配置项。
setPlayListener	设置推流回调接口。

播放基础接口

API	描述
setPlayerView	设置播放器的视频渲染 View。
startPlay	播放器开始播放。
stopPlay	停止播放。
isPlaying	是否正在播放。
pause	暂停播放。
resume	恢复播放。
setSurface	使用 Surface 模式用于本地渲染。
setSurfaceSize	设置渲染 Surface 的大小。

播放配置接口

API	描述
setRenderMode	设置播放渲染模式。
setRenderRotation	设置图像渲染角度。
enableHardwareDecode	开启硬件加速。
setMute	设置是否静音播放。
setAudioRoute	设置声音播放模式。
setVolume	设置音量。
switchStream	多清晰度切换。
setAudioVolumeEvaluationListener	设置音量大小回调接口。

本地录制和截图

API	描述
setVideoRecordListener	设置录制回调接口。
startRecord	启动视频录制。
stopRecord	停止视频录制。
snapshot	播放过程中本地截图。

自定义数据处理

API	描述
addVideoRawData	设置软解码数据载体 Buffer。
setVideoRawDataListener	设置软解码视频数据回调。
setAudioRawDataListener	设置音频数据回调。

直播时移接口

API	描述
prepareLiveSeek	直播时移准备。
seek	直播时移跳转。
resumeLive	恢复直播播放。

截图回调接口类

请参见 ITXSnapshotListener。

API	描述
onSnapshot	截图回调。

软解视频数据回调接口类

请参见 ITXVideoRawDataListener。

API	描述
onVideoRawDataAvailable	软解码器解出一帧数据回调一次。

音频原始数据接口类

请参见 ITXAudioRawDataListener。

API	描述
onPcmDataAvailable	音频播放数据回调，数据格式： PCM。
onAudioInfoChanged	音频播放信息回调。

播放器音量大小接口类

请参见 ITXAudioVolumeEvaluationListener。

API	描述
onAudioVolumeEvaluationNotify	播放器音量大小回调, 取值范围 [0，100]。

TXLivePlayConfig

腾讯云直播播放器的参数配置模块

请参见 TXLivePlayConfig。

主要负责 TXLivePlayer 对应的参数设置，其中绝大多数设置项在播放开始之后再设置是无效的。

常用设置项

API	描述
setAutoAdjustCacheTime	设置是否自动调整缓存时间。

API	描述
setCacheTime	设置播放器缓存时间。
setMaxAutoAdjustCacheTime	设置最大的缓存时间。
setMinAutoAdjustCacheTime	设置最小的缓存时间。
setVideoBlockThreshold	设置播放器视频卡顿报警阈值。
setConnectRetryCount	设置播放器重连次数。
setConnectRetryInterval	设置播放器重连间隔。

专业设置项

API	描述
setEnableMessage	开启消息通道。
enableAEC	设置回声消除。

ITXLivePlayListener

腾讯云直播播放的回调通知

请参见 ITXLivePlayListener。

API	描述
onPlayEvent	播放事件通知。
onNetStatus	网络状态通知。

点播场景
接入文档

最近更新时间：2022-07-06 15:09:19

准备工作

1. 为了您体验到更完整全面的播放器功能，建议您开通 云点播 相关服务，未注册用户可注册账号 试用。若您不使用云点播服务，可略过此步骤，但集成后仅可使用播放器基础能力。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

通过本文你可以学会

- 如何集成腾讯云视立方 Android 播放器SDK
- 如何使用播放器 SDK 进行点播播放
- 如何使用播放器 SDK 底层能力实现更多功能

SDK集成

步骤1：下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

步骤2：配置 License 授权

若您已获得相关 License 授权，需在 腾讯云视立方控制台 获取 License URL 和 License Key：

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 Application 类中进行如下设置：

```
public class MApplication extends Application {

@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
TXLiveBase.setListener(new TXLiveBaseListener() {
@Override
public void onLicenceLoaded(int result, String reason) {
Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
}
});
}
}
```

步骤3: 添加 View

SDK 默认提供 TXCloudVideoView 用于视频渲染，我们第一步要做的就是在布局 xml 文件里加入如下一段代码：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/video_view"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_centerInParent="true"
android:visibility="gone"/>
```

步骤4：创建 Player

接下来创建一个 **TXVodPlayer** 的对象，并使用 setPlayerView 接口将它与我们刚添加到界面上的 **video_view** 控件进行关联。

```
//mPlayerView 即步骤3中添加的视频渲染 view
TXCloudVideoView mPlayerView = findViewById(R.id.video_view);
//创建 player 对象
TXVodPlayer mVodPlayer = new TXVodPlayer(getActivity());
//关联 player 对象与视频渲染 view
mVodPlayer.setPlayerView(mPlayerView);
```

步骤5：启动播放

TXVodPlayer 支持两种播放模式，您可以根据需要自行选择：

通过 URL 方式

TXVodPlayer 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startPlay 函数即可。

```
// 播放 URL 视频资源
String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
mVodPlayer.startPlay(url);


// 播放本地视频资源
String localFile = "/sdcard/video.mp4";
mVodPlayer.startPlay(localFile);
```

通过 FileId 方式

```
// 推荐使用下面的新接口
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // 腾讯云账户的appId
"4564972819220421305", // 视频的fileId
"psignxxxxxxx"); // 如果是加密视频，必须填写
mVodPlayer.startPlay(playInfoParam);

// 旧接口，不推荐使用
TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
authBuilder.setAppId(1252463788);
authBuilder.setFileId("4564972819220421305");
mVodPlayer.startPlay(authBuilder);
```

在 媒资管理 找到对应的视频文件。在文件名下方可以看到 FileId。

通过 FileId 方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或 FileId 不存在，则会收到 TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL 事件，反之收到 TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC 表示请求成功。

步骤6：结束播放

结束播放时**记得销毁 view 控件**，尤其是在下次 startPlay 之前，否则会产生大量的内存泄露以及闪屏问题。

同时，在退出播放界面时，记得一定要调用渲染 View 的 onDestroy() 函数，否则可能会产生内存泄露和“Receiver not registered”报警。

```
@Override
public void onDestroy() {
super.onDestroy();
mVodPlayer.stopPlay(true); // true 代表清除最后一帧画面
mPlayerView.onDestroy();
}
```

> 说明：
> stopPlay 的布尔型参数含义为：“是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。

如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

基础功能使用

1、播放控制

开始播放

```
// 开始播放
mVodPlayer.startPlay(url)
```

暂停播放

```
// 暂停播放
mVodPlayer.pause();
```

恢复播放

```
// 恢复播放
mVodPlayer.resume();
```

结束播放

```
// 结束播放
mVodPlayer.stopPlay(true);
```

调整进度（Seek）

当用户拖拽进度条时，可调用 seek 从指定位置开始播放，播放器 SDK 支持精准 seek。

```
int time = 600; // int类型时，单位为 秒
// float time = 600; // float 类型时单位为 秒
// 调整进度
mVodPlayer.seek(time);
```

从指定时间开始播放

首次调用 startPlay 之前，支持从指定时间开始播放。

```
float startTimeInMS = 600; // 单位：毫秒
mVodPlayer.setStartTime(startTimeInMS); // 设置开始播放时间
mVodPlayer.startPlay(url);
```

2、画面调整

- **view：大小和位置**
  如需修改画面的大小及位置，直接调整 SDK 集成时 添加 View 中添加的 “video_view” 控件的大小和位置即可。
- **setRenderMode：铺满或适应**

可选值	含义
RENDER_MODE_FULL_FILL_SCREEN	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全
RENDER_MODE_ADJUST_RESOLUTION	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边

- **setRenderRotation：画面旋转**

可选值	含义
RENDER_ROTATION_PORTRAIT	正常播放（Home 键在画面正下方）
RENDER_ROTATION_LANDSCAPE	画面顺时针旋转270度（Home 键在画面正左方）

```
// 将图像等比例铺满整个屏幕
mVodPlayer.setRenderMode(TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN);
```

```
// 正常播放（Home 键在画面正下方）
mVodPlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
```

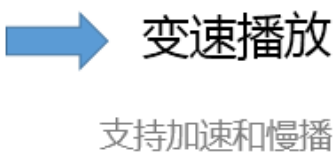
最长边填充

完全填充

横屏模式

3、变速播放

点播播放器支持变速播放，通过接口 setRate 设置点播播放速率来完成，支持快速与慢速播放，如0.5X、1.0X、1.2X、2X等。



```
// 设置1.2倍速播放
mVodPlayer.setRate(1.2);
```

4、循环播放

```
// 设置循环播放
mVodPlayer.setLoop(true);
// 获取当前循环播放状态
mVodPlayer.isLoop();
```

5、静音设置

```
// 设置静音，true 表示开启静音， false 表示关闭静音
mVodPlayer.setMute(true);
```

6、屏幕截图

通过调用 snapshot 您可以截取当前视频为一帧画面，此功能只会截取当前直播流的视频画面，如果您需要截取当前的整个 UI 界面，请调用 Android 的系统 API 来实现。


```
// 屏幕截图
mVodPlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
```

```
//获取到截图bitmap
}
}
});
```

7、贴片广告

播放器SDK支持在界面添加图片贴片，用于广告宣传等。实现方式如下：

- 将autoPlay为 NO，此时播放器会正常加载，但视频不会立刻开始播放。
- 在播放器加载出来后、视频尚未开始时，即可在播放器界面查看图片贴片广告。
- 待达到广告展示结束条件时，使用resume接口启动视频播放。

```
mVodPlayer.setAutoPlay(false); // 设置为非自动播放
mVodPlayer.startPlay(url); // 调用startPlay 后会加载视频，加载成功后不会自动播放
// ......
// 在播放器界面上展示广告
// ......
mVodPlayer.resume(); // 广告展示完调用 resume 开始播放视频
```

8、HTTP-REF

TXVodPlayConfig 中的 headers 可以用来设置 HTTP 请求头，例如常用的防止 URL 被到处拷贝的 Referer 字段（腾讯云可以提供更加安全的签名防盗链方案），以及用于验证客户端身份信息的 Cookie 字段。

9、硬件加速

对于蓝光级别（1080p）的画质，简单采用软件解码的方式很难获得较为流畅的播放体验，所以如果您的场景是以游戏直播为主，一般都推荐开启硬件加速。

软解和硬解的切换需要在切换之前先 **stopPlay**，切换之后再 **startPlay**，否则会产生比较严重的花屏问题。

```
mVodPlayer.stopPlay(true);
mVodPlayer.enableHardwareDecode(true);
mVodPlayer.startPlay(flvUrl, type);
```

10、清晰度设置

SDK 支持 HLS 的多码率格式，方便用户切换不同码率的播放流，从而达到播放不同清晰的目标。在收到 PLAY_EVT_PLAY_BEGIN 事件后，可以通过下面方法进行清晰度设置。

```
ArrayList<TXBitrateItem> bitrates = mVodPlayer.getSupportedBitrates(); //获取多码率数组
int index = bitrates.get(i).index; // 指定要播的码率下标
mVodPlayer.setBitrateIndex(index); // 切换码率到想要的清晰度
```

在播放过程中，可以随时通过 mVodPlayer.setBitrateIndex(int) 切换码率。切换过程中，会重新拉取另一条流的数据，SDK 针对腾讯云的多码率文件做过优化，可以做到切换无卡顿。

11、码流自适应

SDK 支持 HLS 的多码流自适应，开启相关能力后播放器能够根据当前带宽，动态选择最合适的码率播放。在收到 PLAY_EVT_PLAY_BEGIN 事件后，可以通过下面方法开启码流自适应。

```
mVodPlayer.setBitrateIndex(-1); //index 参数传入-1
```

在播放过程中，可以随时通过 mVodPlayer.setBitrateIndex(int) 切换其它码率，切换后码流自适应也随之关闭。

12、播放进度监听

点播播放中的进度信息分为2种：**加载进度**和**播放进度**，SDK 目前是以事件通知的方式将这两个进度实时通知出来的。更多事件通知内容参见事件监听。

您可以为 TXVodPlayer 对象绑定一个 **TXVodPlayerListener** 监听器，进度通知会通过 **PLAY_EVT_PLAY_PROGRESS** 事件回调到您的应用程序，该事件的附加信息中即包含上述两个进度指标。

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
if (event == TXLiveConstants.PLAY_EVT_PLAY_PROGRESS) {
// 加载进度，单位是毫秒
int playable_duration_ms = param.getInt(TXLiveConstants.EVT_PLAYABLE_DURATION_MS);
mLoadBar.setProgress(playable_duration_ms); // 设置loading 进度条

// 播放进度，单位是毫秒
int progress_ms = param.getInt(TXLiveConstants.EVT_PLAY_PROGRESS_MS);
mSeekBar.setProgress(progress_ms); // 设置播放进度条

// 视频总长，单位是毫秒
int duration_ms = param.getInt(TXLiveConstants.EVT_PLAY_DURATION_MS);
// 可以用于设置时长显示等等
}
}

@Override
public void onNetStatus(TXVodPlayer player, Bundle bundle) {
}
});
```

13、播放网速监听

通过 事件监听 方式，可以在视频播放卡顿时在显示当前网速。

- **通过onNetStatus的NET_STATUS_NET_SPEED获取当前网速。具体使用方法见 状态反馈（onNetStatus）。**
- **监听到PLAY_EVT_PLAY_LOADING事件后，显示当前网速。**
- **收到PLAY_EVT_VOD_LOADING_END事件后，对显示当前网速的 view 进行隐藏。**

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
if (event == TXLiveConstants.PLAY_EVT_PLAY_LOADING) {
// 显示当前网速

} else if (event == TXLiveConstants.PLAY_EVT_VOD_LOADING_END) {
// 对显示当前网速的 view 进行隐藏
```

```
}
}

@Override
public void onNetStatus(TXVodPlayer player, Bundle bundle) {
// 获取实时速率, 单位：kbps
int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
}
});
```

14、获取视频分辨率

播放器 SDK 通过 URL 字符串播放视频，URL 中本身不包含视频信息。为获取相关信息，需要通过访问云端服务器加载到相关视频信息，因此 SDK 只能以事件通知的方式将视频信息发送到您的应用程序中，更多内容参见 事件监听。

可以通过下面两种方法获取分辨率信息

方法1：通过 onNetStatus 的 NET_STATUS_VIDEO_WIDTH 和 NET_STATUS_VIDEO_HEIGHT 获取视频的宽和高。具体使用方法见 状态反馈（onNetStatus）。

方法2：在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后，直接调用 TXVodPlayer.getWidth() 和 TXVodPlayer.getHeight() 获取当前宽高。

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
}

@Override
public void onNetStatus(TXVodPlayer player, Bundle bundle) {
//获取视频宽度
int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
//获取视频高度
int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
}
});

// 获取视频宽高，需要在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后才返回值
mVodPlayer.getWidth();
mVodPlayer.getHeight();
```

15、播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10); // 播放时最大缓冲大小。单位：MB
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

16、视频本地缓存

在短视频播放场景中，视频文件的本地缓存是很刚需的一个特性，对于普通用户而言，一个已经看过的视频再次观看时，不应该再消耗一次流量。

- **格式支持：**SDK 支持 HLS(m3u8) 和 MP4 两种常见点播格式的缓存功能。
- **开启时机：**SDK 并不默认开启缓存功能，对于用户回看率不高的场景，也并不推荐您开启此功能。
- **开启方式：**全局生效，在使用播放器开启。开启此功能需要配置两个参数：本地缓存目录及缓存大小。

```
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
if (sdcardDir != null) {
//设置播放引擎的全局缓存目录
TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
//设置播放引擎的全局缓存目录和缓存大小，//单位MB
TXPlayerGlobalSetting.setMaxCacheSize(200);
}

//使用播放器
```

> 说明：
> 旧版本通过 TXVodPlayConfig#setMaxCacheItems 接口配置已经废弃，不推荐使用。

高级功能使用

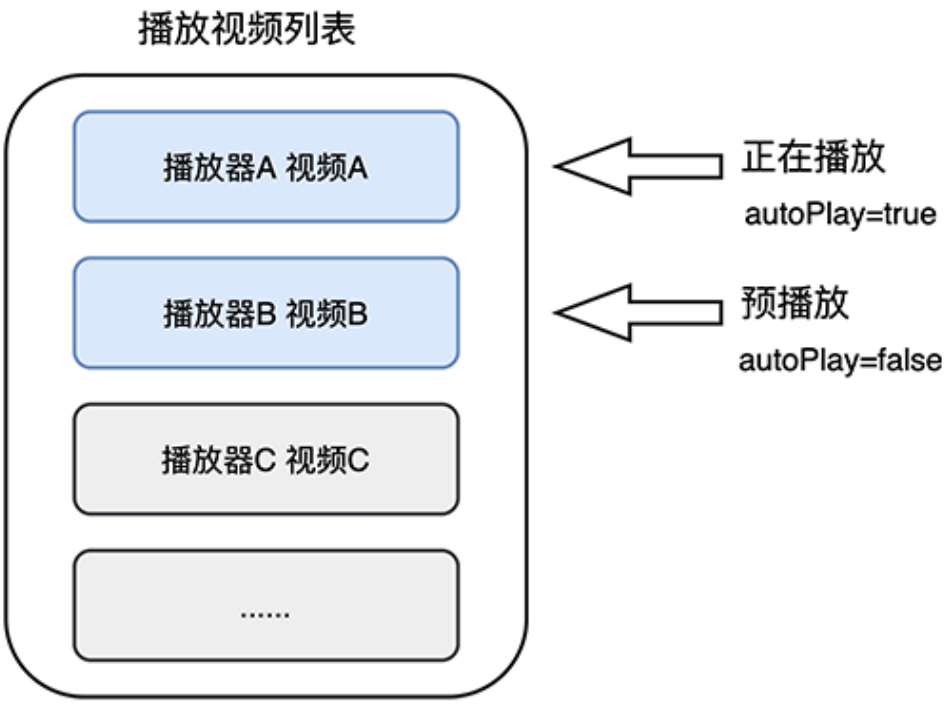
1、视频预播放

步骤1：视频预播放使用

在短视频播放场景中，视频预播放功能对于流畅的观看体验很有帮助：在观看当前视频的同时，在后台加载即将要播放的下一个视频 ，这样一来，当用户真正切换到下一个视频时，已经不需要从头开始加载了，而是可以做到立刻播放。

预播放视频会有很好的秒开效果，但有一定的性能开销，如果业务同时有较多的视频预加载需求，建议结合 视频预下载 一起使用。

这就是视频播放中无缝切换的背后技术支撑，您可以使用 TXVodPlayer 中的 setAutoPlay 开关来实现这个功能，具体做法如下：




```
// 播放视频 A: 如果将 autoPlay 设置为 true， 那么 startPlay 调用会立刻开始视频的加载和播放
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
playerA.setAutoPlay(true);
playerA.startPlay(urlA);

// 在播放视频 A 的同时，预加载视频 B，做法是将 setAutoPlay 设置为 false
String urlB = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
playerB.setAutoPlay(false);
playerB.startPlay(urlB); // 不会立刻开始播放，而只会开始加载视频
```

等到视频 A 播放结束，自动（或者用户手动切换到）视频 B 时，调用 resume 函数即可实现立刻播放。

> ⚠ **注意：**
> 设置了 autoPlay 为 false 之后，调用 resume 之前需要保证视频 B 已准备完成，即需要在监听到视频 B 的 PLAY_EVT_VOD_PLAY_PREPARED（2013，播放器已准备完成，可以播放）事件后调用。

```
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
// 在视频 A 播放结束的时候，直接启动视频 B 的播放，可以做到无缝切换
```

```
if (event == PLAY_EVT_PLAY_END) {
playerA.stop();
playerB.setPlayerView(mPlayerView);
playerB.resume();
}
}
```

步骤2：视频预播放缓冲配置

- 设置较大的缓冲可以更好的应对网络的波动，达到流畅播放的目的。
- 设置较小的缓冲可以帮助节省流量消耗。

预播放缓冲大小

此接口针对预加载场景（即在视频启播前，且设置 player 的 AutoPlay 为 false），用于控制启播前阶段的最大缓冲大小。

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxPreloadSize(2); // 预播放最大缓冲大小。单位：MB, 根据业务情况设置去节省流量
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10); // 播放时最大缓冲大小。单位：MB
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

2、视频预下载

不需要创建播放器实例，预先下载视频部分内容，使用播放器时，可以加快视频启播速度，提供更好的播放体验。

在使用播放服务前，请确保先设置好 视频缓存。

> 说明：
>
> 1. TXPlayerGlobalSetting 是全局缓存设置接口，原有 TXVodConfig 的缓存配置接口废弃。
> 2. 全局缓存目录和大小设置的优先级高于播放器 TXVodConfig 配置的缓存设置。

使用示例：

```
//先设置播放引擎的全局缓存目录和缓存大小
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
//设置播放引擎的全局缓存目录和缓存大小
if (sdcardDir != null) {
TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
TXPlayerGlobalSetting.setMaxCacheSize(200); //单位MB
}

String palyrl = "http://****";
//启动预下载
final TXVodPreloadManager downloadManager = TXVodPreloadManager.getInstance(getApplicationC
ontext());
final int taskID = downloadManager.startPreload(playUrl, 3, 1920*1080, new ITXVodPreloadListener()
{
@Override
public void onComplete(int taskID, String url) {
Log.d(TAG, "preload: onComplete: url: " + url);
}

@Override
public void onError(int taskID, String url, int code, String msg) {
Log.d(TAG, "preload: onError: url: " + url + ", code: " + code + ", msg: " + msg);
}

});

//取消预下载
downloadManager.stopPreload(taskID);
```

3、视频下载

视频下载支持用户在有网络的条件下下载视频，随后在无网络的环境下观看。同时播放器 SDK 提供本地加密能力，下载后的本地视频仍为加密状态，仅可通过指定播放器对视频进行解密播放，可有效防止下载后视频的非法传播，保护视频安全。

由于 HLS 流媒体无法直接保存到本地，因此也无法通过播放本地文件的方式实现 HLS 下载到本地后播放，对于该问题，您可以通过基于 TXVodDownloadManager 的视频下载方案实现 HLS 的离线播放。

> ⚠ 注意：

- TXVodDownloadManager 暂不支持缓存 MP4 和 FLV 格式的文件，仅支持缓存非嵌套 HLS 格式文件。
- 播放器 SDK 已支持 MP4 和 FLV 格式的本地文件播放。

步骤1：准备工作

TXVodDownloadManager 被设计为单例，因此您不能创建多个下载对象。用法如下：

```
TXVodDownloadManager downloader = TXVodDownloadManager.getInstance();
downloader.setDownloadPath("<指定您的下载目录>");
```

步骤2: 开始下载

开始下载有 URL 和 Fileid 两种方式，具体操作如下：

URL 方式

至少需要传入下载地址 URL，不支持嵌套 HLS 格式，仅支持非嵌套 HLS 格式下载。userName 不传入具体值时，默认为"default"。

```
downloader.startDownloadUrl("http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq1500005830/00eb06a886022680114373569 84/video_10_0.m3u8", "");
```

Fileid 方式

Fileid 下载至少需要传入 AppID、 Fileid 和 qualityId。带签名视频需传入 pSign，userName 不传入具体值时，默认为“default”。

```
// QUALITYOD // 原画
// QUALITYFLU // 流畅
// QUALITYSD // 标清
// QUALITYHD // 高清
TXVodDownloadDataSource source = new TXVodDownloadDataSource(1252463788, "4564972819220421305", QUALITYHD, "", "");
downloader.startDownload(source);
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 listener。

```
downloader.setListener(this);
```

可能收到的任务回调有：

回调信息	说明
void onDownloadStart(TXVodDownloadMediaInfo mediaInfo)	任务开始，表示 SDK 已经开始下载
void onDownloadProgress(TXVodDownloadMediaInfo mediaInfo)	任务进度，下载过程中，SDK 会频繁回调此接口，您可以通过mediaInfo.getProgress() 获取当前进度
void onDownloadStop(TXVodDownloadMediaInfo mediaInfo)	任务停止，当您调用 stopDownload 停止下载，收到此消息表示停止成功
void onDownloadFinish(TXVodDownloadMediaInfo mediaInfo)	下载完成，收到此回调表示已全部下载。此时下载文件可以给 TXVodPlayer 播放
void onDownloadError(TXVodDownloadMediaInfo mediaInfo, int error, String reason)	下载错误，下载过程中遇到网络断开会回调此接口，同时下载任务停止。错误码位于TXVodDownloadManager中

由于 downloader 可以同时下载多个任务，所以回调接口里带上了 TXVodDownloadMediaInfo 对象，您可以访问 URL 或 dataSource 判断下载源，同时还可以获取到下载进度、文件大小等信息。

步骤4：中断下载

停止下载请调用 downloader.stopDownload() 方法，参数为 downloader.startDownload() 返回的对象。SDK 支持断点续传，当下载目录没有发生改变时，下次下载同一个文件时会从上次停止的地方重新开始。

步骤5：管理下载

获取所有用户账户的下载列表信息, 也可获取指定用户账户的下载列表信息。

```
// 获取所有用户的下载列表信息
// 接入方可根据下载信息中的userName区分不同用户的下载列表信息
List<TXVodDownloadMediaInfo> downloadList = downloader.getDownloadMediaInfoList();
if (downloadInfoList == null || downloadInfoList.size() <= 0) return;
// 获取默认“default”用户的下载列表
List<TXVodDownloadMediaInfo> defaultUserDownloadList = new ArrayList<>();
for(TXVodDownloadMediaInfo downloadMediaInfo : downloadInfoList) {
if ("default".equals(downloadMediaInfo.getUserName())) {
defaultUserDownloadList.add(downloadMediaInfo);
}
}
```

获取某个 Fileid 相关下载信息，包括当前下载状态，获取当前下载进度，判断是否下载完成等，需要传入 AppID、Fileid 和 qualityId。

```
// 获取某个fileId相关下载信息
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo(1252463788, "4564972819220421305", QUALITYHD);
int duration = downloadInfo.getDuration(); // 获取总时长
int playableDuration = downloadInfo.getPlayableDuration(); // 获取已下载的可播放时长
float progress = downloadInfo.getProgress(); // 获取下载进度
String playPath = downloadInfo.getPlayPath(); // 获取离线播放路径，传给播放器即可离线播放
int downloadState = downloadInfo.getDownloadState(); // 获取下载状态，具体参考STATE_xxx常量
boolean isDownloadFinished = downloadInfo.isDownloadFinished(); // 返回true表示下载完成
```

获取某个 URL 相关下载信息，需要传入 URL 信息。

```
// 获取某个url下载信息
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo("http://1253131631.vod2.myqcloud.com/26f327f9vodgzp1253131631/f4bdff799031868222924043041/playlist.m3u8”);
```

删除下载信息和相关文件，需传入 TXVodDownloadMediaInfo 参数。

```
// 删除下载信息
boolean deleteRst = downloader.deleteDownloadMediaInfo(downloadInfo);
```

4、加密播放

视频加密方案主要用于在线教育等需要对视频版权进行保护的场景。如果要对您的视频资源进行加密保护，就不仅需要在播放器上做改造，还需要对视频源本身进行加密转码，亦需要您的后台和终端研发工程师都参与其中。在 视频加密解决方案 中您会了解到全部细节内容。

5、播放器配置

在调用 statPlay 之前可以通过 setConfig 对播放器进行参数配置，例如：设置播放器连接超时时间、设置进度回调间隔、设置缓存文件个数等配置，TXVodPlayConfig 支持配置的详细参数请单击 基础配置接口 了解。使用示例：

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setEnableAccurateSeek(true); // 设置是否精确 seek，默认 true
config.setMaxCacheItems(5); // 设置缓存文件个数为5
config.setProgressInterval(200); // 设置进度回调间隔，单位毫秒
```

```
config.setMaxBufferSize(50); // 最大预加载大小，单位 MB
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

启播时指定分辨率

播放 HLS 的多码率视频源，如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率。 播放器会查找小于或等于偏好分辨率的流进行启播，启播后没有必要再通过 setBitrateIndex 切换到需要的码流。

```
TXVodPlayConfig config = new TXVodPlayConfig();
// 传入参数为视频宽和高的乘积(宽 * 高），可以自定义值传入
config.setPreferredResolution(TXLiveConstants.VIDEO_RESOLUTION_720X1280);
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

设置播放进度回调时间间隔

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setProgressInterval(200); // 设置进度回调间隔，单位毫秒
mVodPlayer.setConfig(config); // 把config 传给 mVodPlayer
```

播放器事件监听

您可以为 TXVodPlayer 对象绑定一个 TXVodPlayListener 监听器，即可通过onPlayEvent（事件通知）和 onNetStatus（状态反馈）向您的应用程序同步信息。

播放事件通知（onPlayEvent）

事件 ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度 和总体时长
PLAY_EVT_PLAY_LOADING	2007	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件
PLAY_EVT_VOD_LOADING_END	2014	视频播放 loading 结束，视频继续播放
TXVodConstants.VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seek 完成，10.3版本开始支持

结束事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败

警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件 ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解

连接事件

连接服务器的事件，主要用于测定和统计服务器连接时间：

事件 ID	数值	含义说明
PLAY_EVT_VOD_PLAY_PREPARED	2013	播放器已准备完成，可以播放。设置了 autoPlay 为 false 之后，需要在收到此事件后，调用 resume 才会开始播放
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包（IDR）

画面事件

以下事件用于获取画面变化信息：

事件 ID	数值	含义说明
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率改变
PLAY_EVT_CHANGE_ROTATION	2011	MP4 视频旋转角度

视频信息事件

事件 ID	数值	含义说明
TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC	2010	成功获取播放文件信息

如果通过 fileId 方式播放且请求成功（接口：startPlay(TXPlayerAuthBuilder authBuilder)），SDK 会将一些请求信息通知到上层。您可以在收到 TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC 事件后，解析 param 获取视频信息。

视频信息	含义说明
EVT_PLAY_COVER_URL	视频封面地址
EVT_PLAY_URL	视频播放地址
EVT_PLAY_DURATION	视频时长
EVT_TIME	事件发生时间
EVT_UTC_TIME	UTC 时间
EVT_DESCRIPTION	事件说明
EVT_PLAY_NAME	视频名称
TXVodConstants.EVT_IMAGESPRIT_WEBVTTURL	雪碧图 web vtt 描述文件下载 URL，10.2 版本开始支持
TXVodConstants.EVT_IMAGESPRIT_IMAGEURL_LIST	雪碧图图片下载URL，10.2版本开始支持
TXVodConstants.EVT_DRM_TYPE	加密类型，10.2版本开始支持

通过 onPlayEvent 获取视频播放过程信息示例：

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
// 收到播放器已经准备完成事件，此时可以调用pause、resume、getWidth、getSupportedBitrates 等接口
} else if (event == TXLiveConstants.PLAY_EVT_PLAY_BEGIN) {
// 收到开始播放事件
} else if (event == TXLiveConstants.PLAY_EVT_PLAY_END) {
// 收到开始结束事件
}
}
```

```
@Override
public void onNetStatus(TXVodPlayer player, Bundle bundle) {
}
});
```

播放状态反馈（onNetStatus）

状态反馈每0.5秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前 SDK 内部的一些具体情况，以便您能对当前视频播放状态等有所了解。

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时 CPU 使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 – 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 – 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_VIDEO_CACHE	缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器 IP

通过 onNetStatus 获取视频播放过程信息示例：

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
}

@Override
public void onNetStatus(TXVodPlayer player, Bundle bundle) {
//获取当前CPU使用率
CharSequence cpuUsage = bundle.getCharSequence(TXLiveConstants.NET_STATUS_CPU_USAGE);
//获取视频宽度
int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
```

```
//获取视频高度
int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
//获取实时速率，单位：kbps
int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
//获取当前流媒体的视频帧率
int fps = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_FPS);
//获取当前流媒体的视频码率，单位 kbps
int videoBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_BITRATE);
//获取当前流媒体的音频码率，单位 kbps
int audioBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_AUDIO_BITRATE);
//获取缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
int jitterbuffer = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_CACHE);
//获取连接的服务器的IP地址
String ip = bundle.getString(TXLiveConstants.NET_STATUS_SERVER_IP);
}
});
```

场景化功能

1、基于 SDK 的 Demo 组件

基于播放器 SDK，腾讯云研发了一款 播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美市面上各种流行视频 App 的播放软件。

2、开源 Github

基于播放器 SDK，腾讯云研发了沉浸式视频播放器组件、视频 Feed 流、多播放器复用组件等，而且随着版本发布，我们会提供跟多的基于用户场景的组件。您可以通过 Player_Android 下载体验。

API 文档

最近更新时间：2022-06-01 10:04:45

TXVodPlayer

点播播放器

请参见 TXVodPlayer。

主要负责从指定的点播流地址拉取音视频数据，并进行解码和本地渲染播放。

播放器包含如下能力：

- 支持 FLV、MP4 及 HLS 多种播放格式，支持 基础播放（URL 播放）和 点播播放（Fileid 播放）两种播放方式 。
- 屏幕截图，可以截取当前播放流的视频画面。
- 通过手势操作，调节亮度、声音、进度等。
- 可以手动切换不同的清晰度，也可根据网络带宽自适应选择清晰度。
- 可以指定不同倍速播放，并开启镜像和硬件加速。
- 完整能力，请参见 点播超级播放器 - 能力清单。

播放器配置接口

API	描述
setConfig	设置播放器配置信息，配置信息请参见 TXVodPlayConfig 。
setPlayerView	设置播放器的视频渲染 TXCloudVideoView。
setPlayerView	设置播放器的视频渲染 TextureView。
setSurface	设置播放器的视频渲染 SurfaceView。
setStringOption	设置播放器业务参数，参数格式为<String,Object>。

播放基础接口

API	描述
startPlay	播放 HTTP URL 形式地址。
startPlay	以 fileId 形式播放，传入TXPlayInfoParams参数。
stopPlay	停止播放。
isPlaying	是否正在播放。
pause	暂停播放，停止获取流数据,保留最后一帧画面。

API	描述
resume	恢复播放，重新获取流数据。
seek	跳转到视频流指定时间点，单位秒。
seek	跳转到视频流指定时间点，单位毫秒。
getCurrentPlaybackTime	获取当前播放位置，单位秒。
getBufferDuration	获取缓存的总时长，单位秒。
getDuration	获取总时长，单位秒。
getPlayableDuration	获取可播放时长，单位秒。
getWidth	获取视频宽度。
getHeight	获取视频高度。
setAutoPlay	设置点播是否 startPlay 后自动开始播放，默认自动播放。
setStartTime	设置播放开始时间。
setToken	加密 HLS 的 token。
setLoop	设置是否循环播放。
isLoop	返回是否循环播放状态。

视频相关接口

API	描述
enableHardwareDecode	启用或禁用视频硬解码。
snapshot	获取当前视频帧图像。 **注意：由于获取当前帧图像是比较耗时的操作，所以截图会通过异步回调出来。**
setMirror	设置镜像。
setRate	设置点播的播放速率，默认1.0。
getBitrateIndex	返回当前播放的码率索引。
setBitrateIndex	设置当前正在播放的码率索引，无缝切换清晰度。清晰度切换可能需要等待一小段时间。
setRenderMode	设置 图像平铺模式。
setRenderRotation	设置 图像渲染角度。

音频相关接口

API	描述
setMute	设置是否静音播放。
setAudioPlayoutVolume	设置音量大小，范围：0 – 100。
setRequestAudioFocus	设置是否自动获取音频焦点 默认自动获取。

事件通知接口

API	描述
setPlayListener	设置播放器的回调（已弃用，建议使用 setVodListener ）。
setVodListener	设置播放器的回调。
onNotifyEvent	点播播放事件通知。
onNetSuccess	点播播放网络状态通知。
onNetFailed	播放 fileId 网络异常通知。

TRTC 相关接口

通过以下接口，可以把点播播放器的音视频流通过 TRTC 进行推送，更多 TRTC 服务请参见 TRTC 产品概述。

API	描述
attachTRTC	点播绑定到 TRTC 服务。
detachTRTC	点播解绑 TRTC 服务。
publishVideo	开始推送视频流。
unpublishVideo	取消推送视频流。
publishAudio	开始推送音频流。
unpublishAudio	取消推送音频流。

ITXVodPlayListener

腾讯云点播回调通知。

SDK 基础回调

API	描述
onPlayEvent	点播播放事件通知，请参见 播放事件列表、事件参数。
onNetStatus	点播播放器 网络状态通知。

TXVodPlayConfig

点播播放器配置类。

基础配置接口

API	描述
setConnectRetryCount	设置播放器重连次数。
setConnectRetryInterval	设置播放器重连间隔，单位秒。
setTimeout	设置播放器连接超时时间，单位秒。
setCacheFolderPath	设置点播缓存目录，点播 MP4、HLS 有效。
setMaxCacheItems	设置缓存文件个数。接口废弃，请使用 TXPlayerGlobalSetting#setMaxCacheSize 进行全局配置。
setPlayerType	设置播放器类型。
setHeaders	设置自定义 HTTP headers。
setEnableAccurateSeek	设置是否精确 seek，默认 true。
setAutoRotate	播放 MP4 文件时，若设为 YES 则根据文件中的旋转角度自动旋转。旋转角度可在 PLAY_EVT_CHANGE_ROTATION 事件中获得。默认 YES。
setSmoothSwitchBitrate	平滑切换多码率 HLS，默认 false。
setCacheMp4ExtName	缓存 MP4 文件扩展名。
setProgressInterval	设置进度回调间隔，单位毫秒。
setMaxBufferSize	最大预加载大小，单位 MB。
setMaxPreloadSize	设置预加载最大缓冲大小，单位：MB。
setExtInfo	设置拓展信息。

API	描述
setPreferredResolution	播放 HLS 有多条码流时，根据设定的 preferredResolution 选最优的码流进行起播，preferredResolution 是宽高的乘积（width * height），启播前设置才有效。

TXPlayerGlobalSetting

点播播放器全局配置类

API	描述
setCacheFolderPath	设置播放引擎的cache目录。设置后，离线下载，预下载，播放器等会优先从此目录读取和存储
setMaxCacheSize	设置播放引擎的最大缓存大小。设置后会根据设定值自动清理Cache目录的文件。单位MB。

TXVodPreloadManager

点播播放器预下载接口类

API	描述
getInstance	获取 TXVodPreloadManager 实例对象，单例模式。
startPreload	启动预下载前，请先设置好播放引擎的缓存目录。 TXPlayerGlobalSetting#setCacheFolderPath 和缓存大小。 TXPlayerGlobalSetting#setMaxCacheSize。
stopPreload	停止预下载。

TXVodDownloadManager

点播播放器视频下载接口类。当前只支持下载非嵌套 m3u8 视频源，对 simpleAES 加密视频源将进行腾讯云私有加密算法加密以提升安全性。

API	描述
getInstance	获取 TXVodDownloadManager 实例对象，单例模式。
setHeaders	设置下载 HTTP 头。
setListener	设置下载回调方法，下载前必须设好。
startDownloadUrl	以 URL 方式开始下载。

API	描述
startDownload	以 fileid 方式开始下载。
stopDownload	停止下载，ITXVodDownloadListener.onDownloadStop 回调时停止成功。
deleteDownloadMediaInfo	删除下载信息。
getDownloadMediaInfoList	获取所有用户的下载列表信息。
getDownloadMediaInfo	获取下载信息。

ITXVodDownloadListener

腾讯云视频下载回调通知。

API	描述
onDownloadStart	下载开始
onDownloadProgress	下载进度更新
onDownloadStop	下载停止
onDownloadFinish	下载结束
onDownloadError	下载过程中遇到错误

TXVodDownloadDataSource

腾讯云视频fildid下载源，下载时作为参数传入。

API	描述
TXVodDownloadDataSource	构造函数，传入 appid，fileid，quality，pSign，username 等参数
getAppId	获取传入的 appid
getFileId	获取传入的 fileid
getPSign	获取传入的 psign
getQuality	获取传入的 quality
getUserName	获取传入的 userName，默认“default”

TXVodDownloadMediaInfo

腾讯云视频下载信息，可获取下载进度，播放链接等等信息

API	描述
getDataSource	fileid下载时获取传入的 fileid 下载源
getDuration	获取下载的总时长
getPlayableDuration	获取已下载的可播放时长
getSize	获取下载文件总大小，只针对 fileid 下载源有效
getDownloadSize	获取已下载文件大小，只针对 fileid 下载源有效
getProgress	获取当前下载进度
getPlayPath	获取当前播放路径，可传给 TXVodPlayer 播放
getDownloadState	获取下载状态
isDownloadFinished	判断是否下载完成

错误码表

常规事件

code	事件定义	含义说明
2004	PLAY_EVT_PLAY_BEGIN	视频播放开始（若有转菊花效果，此时将停止）。
2005	PLAY_EVT_PLAY_PROGRESS	视频播放进度，会通知当前播放进度、加载进度和总体时长。
2007	PLAY_EVT_PLAY_LOADING	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件。
2014	PLAY_EVT_VOD_LOADING_END	视频播放 loading 结束，视频继续播放。
2006	PLAY_EVT_PLAY_END	视频播放结束。
2013	PLAY_EVT_VOD_PLAY_PREPARED	播放器已准备完成，可以播放。
2003	PLAY_EVT_RCV_FIRST_I_FRAME	网络接收到首个可渲染的视频数据包（IDR）。
2009	PLAY_EVT_CHANGE_RESOLUTION	视频分辨率改变。
2011	PLAY_EVT_CHANGE_ROTATION	MP4 视频旋转角度。

警告事件

code	事件定义	含义说明
-2301	PLAY_ERR_NET_DISCONNECT	网络断连，且经多次重连亦不能恢复，更多重试请自行重启播放。
-2305	PLAY_ERR_HLS_KEY	HLS 解密 key 获取失败。
2101	PLAY_WARNING_VIDEO_DECODE_FAIL	当前视频帧解码失败。
2102	PLAY_WARNING_AUDIO_DECODE_FAIL	当前音频帧解码失败
2103	PLAY_WARNING_RECONNECT	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT)。
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解。
-2304	PLAY_ERR_HEVC_DECODE_FAIL	H265 解码失败。
-2303	PLAY_ERR_FILE_NOT_FOUND	播放的文件不存在。