播放器 SDK

不含 UI 的集成方案

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或95716。

文档目录

- 不含 UI 的集成方案
  - Web
    - TCPlayer 集成指引
    - TCPlayer 清晰度配置说明
    - TCPlayer 快直播降级说明
  - iOS
    - SDK 集成指引
    - 点播场景
    - 直播场景
  - Android
    - SDK 集成指引
    - 点播场景
    - 直播场景
  - Flutter
    - SDK 集成指引
    - 点播场景
    - 直播场景

不含 UI 的集成方案
Web
TCPlayer 集成指引

最近更新时间：2024-02-29 09:35:51

本文档将介绍适用于点播播放和直播播放的 Web 播放器 SDK（TCPlayer），它可快速与自有 Web 应用集成，实现视频播放功能。Web 播放器 SDK（TCPlayer）内默认包含部分 UI，您可按需取用。

概述

Web 播放器是通过 HTML5 的 `<video>` 标签以及 Flash 实现视频播放。在浏览器不支持视频播放的情况下，实现了视频播放效果的多平台统一体验，并结合腾讯云点播视频服务，提供防盗链和播放 HLS 普通加密视频等功能。

协议支持

音视频协议	用途	URL 地址格式	PC 浏览器	移动浏览器
MP3	音频	https://xxx.vod.myqcloud.com/xxx.mp3	支持	支持
MP4	点播	https://xxx.vod.myqcloud.com/xxx.mp4	支持	支持
HLS（M3U8）	直播	https://xxx.liveplay.myqcloud.com/xxx.m3u8	支持	支持
	点播	https://xxx.vod.myqcloud.com/xxx.m3u8	支持	支持
FLV	直播	https://xxx.liveplay.myqcloud.com/xxx.flv	支持	部分支持
	点播	https://xxx.vod.myqcloud.com/xxx.flv	支持	部分支持
WebRTC	直播	webrtc://xxx.liveplay.myqcloud.com/live/xxx	支持	支持

说明：

- 视频编码格式仅支持 H.264 编码。
- 播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。
- HLS、FLV 视频在部分浏览器环境播放需要依赖 Media Source Extensions 。

- 在不支持 WebRTC 的浏览器环境，传入播放器的 WebRTC 地址会自动进行协议转换来更好的支持媒体播放。

功能支持

功能\浏览器	Chrome	Firefox	Edge	QQ浏览器	Mac Safari	iOS Safari	微信	Android Chrome	IE11
播放器尺寸设置	✓	✓	✓	✓	✓	✓	✓	✓	✓
续播功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
倍速播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
缩略图预览	✓	✓	✓	✓	–	–	–	–	✓
切换 fileID 播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
镜像功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
进度条标记	✓	✓	✓	✓	✓	–	–	–	✓
HLS 自适应码率	✓	✓	✓	✓	✓	✓	✓	✓	✓
Referer 防盗链	✓	✓	✓	✓	✓	✓	✓	–	✓
清晰度切换提示	✓	✓	✓	✓	–	–	–	✓	✓
试看功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 标准加密播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 私有加密播放	✓	✓	✓	–	–	–	• Android: ✓ • iOS: –	✓	✓

视频统计信息	✓	✓	✓	✓	–	–	–	–	–
视频数据监控	✓	✓	✓	✓	–	–	–	–	–
自定义提示文案	✓	✓	✓	✓	✓	✓	✓	✓	✓
自定义 UI	✓	✓	✓	✓	✓	✓	✓	✓	✓
弹幕	✓	✓	✓	✓	✓	✓	✓	✓	✓
水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
幽灵水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
视频列表	✓	✓	✓	✓	✓	✓	✓	✓	✓
弱网追帧	✓	✓	✓	✓	✓	✓	✓	✓	✓

说明：

- 视频编码格式仅支持 H.264 编码。
- Chrome、Firefox 包括 Windows、macOS 平台。
- Chrome、Firefox、Edge 及 QQ 浏览器播放 HLS 需要加载 `hls.js` 。
- Referer 防盗链功能是基于 HTTP 请求头的 Referer 字段实现的，部分 Android 浏览器发起的 HTTP 请求不会携带 Referer 字段。

播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。例如：在 Chrome 等现代浏览器中优先使用 HTML5 技术实现视频播放，而手机浏览器上会使用 HTML5 技术或者浏览器内核能力实现视频播放。

准备工作

播放器 SDK Web 端（TCPlayer）自 5.0.0 版本起需获取 License 授权后方可使用。若您无需使用新增的高级功能，可直接申请基础版 License 以**继续免费使用 TCPlayer**；若您需要使用新增的高级功能则需购买高级版 License。具体信息如下：

TCPlayer 功能	功能范围	所需 License	定价	授权单位
基础版功能	包含 5.0.0 以前版本提供的全部功能，详情见 产品功能	播放器 Web 端基础版 License	0元 免费申请	精准域名（1个 License 最多可授权10个精准域名）

高级版功能	基础版功能、VR 播放 、安全检查	播放器 Web 端高级版 License	399元/月 立即购买	泛域名（1个 License 最多可授权1个泛域名）

说明：
播放器 Web 端基础版 License 可免费申请，申请后有效期默认1年；在有效期剩余时间小于30天时，可免费续期。

集成指引

通过以下步骤，您就可以在网页上添加一个视频播放器。

步骤1：在页面中引入文件

播放器 SDK 支持 cdn 和 npm 两种集成方式：

1. 通过 cdn 集成

建议在使用播放器 SDK 的时候自行部署资源，单击下载播放器资源 。部署解压后的文件夹，不能调整文件夹里面的目录，避免资源互相引用异常。
如果您部署的地址为 `aaa.xxx.ccc` ，在合适的地方引入播放器样式文件与脚本文件，自行部署情况下，需要手动引用资源包文件夹 libs 下面的依赖文件，否则会默认请求腾讯云 cdn 文件。在 html 页面内引入播放器样式文件与脚本文件：

```
<link href="https://aaa.xxx.ccc/tcplayer.min.css" rel="stylesheet"/>
<!--如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在 tcplayer.vx.x.x.min.js 之前引入 hls.min.x.xx.m.js。-->
<script src="https://aaa.xxx.ccc/libs/hls.min.x.xx.m.js"></script>
<!--播放器脚本文件-->
<script src="https://aaa.xxx.ccc/tcplayer.vx.x.x.min.js"></script>
```

2. 通过 npm 集成

首先安装 tcplayer 的 npm 包：

```
npm install tcplayer.js
```

导入 SDK 和样式文件：

```
import TCPlayer from 'tcplayer.js';
import 'tcplayer.js/dist/tcplayer.min.css';
```

步骤2：放置播放器容器

在需要展示播放器的页面位置加入播放器容器。例如，在 index.html 中加入如下代码（容器 ID 以及宽高都可以自定义）。

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline
webkit-playsinline>
</video>
```

说明：

- 播放器容器必须为 `<video>` 标签。
- 示例中的 `player-container-id` 为播放器容器的 ID，可自行设置。
- 播放器容器区域的尺寸，建议通过 CSS 进行设置，通过 CSS 设置比属性设置更灵活，可以实现例如铺满全屏、容器自适应等效果。
- 示例中的 `preload` 属性规定是否在页面加载后载入视频，通常为了更快的播放视频，会设置为 `auto`，其他可选值：`meta`（当页面加载后只载入元数据），`none`（当页面加载后不载入视频），移动端由于系统限制不会自动加载视频。
- `playsinline` 和 `webkit-playsinline` 这几个属性是为了在标准移动端浏览器不劫持视频播放的情况下实现行内播放，此处仅作示例，请按需使用。
- 设置 `x5-playsinline` 属性在 TBS 内核会使用 X5 UI 的播放器。

步骤3：播放器初始化

页面初始化后，即可播放视频资源。播放器同时支持点播和直播两种播放场景，具体播放方式如下：

- 点播播放：播放器可以通过 FileID 播放腾讯云点播媒体资源，云点播具体流程请参见 使用播放器播放 文档。
- 直播播放：播放器通过传入 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）

在页面初始化之后，调用播放器实例上的方法，将 URL 地址传入方法。

```
// player-container-id 为播放器容器 ID，必须与 html 中一致
var player = TCPlayer('player-container-id', {
    sources: [{
      src: 'path/to/video', // 播放地址
    }],
    licenseUrl: 'license/url', // license 地址，参考准备工作部分，在视立方控制台申请
license 后可获得 licenseUrl
});

// player.src(url); // url 播放地址
```

通过 FileID 播放（点播）

在 index.html 页面初始化的代码中加入以下初始化脚本，传入在准备工作中获取到的 fileID（媒资管理）中的视频 ID 与 appID（在账号信息>基本信息中查看）。

```
var player = TCPlayer('player-container-id', { // player-container-id 为播放器容器 ID，必须与 html 中一致
    fileID: '3701925921299637010', // 请传入需要播放的视频 fileID（必须）
    appID: '1500005696', // 请传入点播账号的 appID（必须）
    //私有加密播放需填写 psign， psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436

psign:'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwNTY5NiwiZmlsZUlkIjoiMzcwMTkyNTkyMTI5OTYzNzAxMCIsImN1cnJlbnRUaW1lU3RhbXAiOjE2MjY4NjAxNzYsImV4cGlyZVRpbWVTdGFtcCI6MjYyNjg1OTE3OSwicGNmZyI6InByaXZhdGUiLCJ1cmxBY2Nlc3NJbmZvIjp7InQiOiI5YzkyYjBhYiJ9LCJkcm1MaWNlbnNlSW5mbyI6eyJleHBpcmVUaW1lU3RhbXAiOjI2MjY4NTkxNzksInN0cmljdE1vZGUiOjJ9fQ.Bo5K5ThInc4n8AIzIZQ-CP9a49M2mEr9-zQLH9ocQgI',
    licenseUrl: 'license/url', // 参考准备工作部分，在视立方控制台申请 license 后可获得 licenseUrl
});
```

注意：

要播放的视频建议使用腾讯云转码，原始视频无法保证在浏览器中正常播放。

步骤4: 更多功能

播放器可以结合云点播的服务端能力实现高级功能，比如自动切换自适应码流、预览视频缩略图、添加视频打点信息等。这些功能在 播放长视频方案 中有详细的说明，可以参考文档实现。
此外，播放器还提供更多其他功能，功能列表和使用方法请参见 功能展示 页面。

TCPlayer 清晰度配置说明

最近更新时间：2023-10-11 15:06:31

概述

在播放过程中，您可以通过自动或手动切换视频清晰度，以适应不同尺寸的播放设备和网络环境，从而提高观看体验。本文将从以下几个场景进行说明。

直播场景

直播场景以 URL 的形式来播放视频，初始化播放器时，可以通过 sources 字段指定所要播放的 URL。或者在初始化播放器之后，调用播放器实例上的 src 方法进行播放。

1. 自适应码率（ABR）

自适应码率地址在切换时可以做到无缝衔接，切换过程不会出现中断或跳变，实现了观感和听感的平滑过渡。使用该技术也比较简单，仅需将播放地址传给播放器，播放器会自动解析子流，并将清晰度切换组件渲染到控制条上。

示例1: 播放 HLS 自适应码率地址

在初始化播放器时，传入自适应码率地址，播放器将自动生成清晰度切换组件，并会根据网络状况进行自动切换。

```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器ID，必须与html中一致
  sources: [{
    src: 'https://hls-abr-url', // hls 自适应码率地址
  }],
```

```
});
```

> **注意：**
> 解析 HLS 自适应码率的子流，需要依赖播放环境的 MSE API。在不支持 MSE 的浏览器环境（例如 iOS 的 Safari 浏览器），会由浏览器内部自动根据网络情况来切换清晰度，但无法解析出多种清晰度来供您手动切换。

示例2：播放 WebRTC 自适应码率地址

在 WebRTC 自适应码率场景，传入地址后，播放器会根据地址中的 ABR 模板自动拆解子流地址。

```
const player = TCPlayer('player-container-id',{
 sources: [{
  src: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=69f1eb8c&tabr_bitrates=d10
80p,d540p,d360p&tabr_start_bitrate=d1080p&tabr_control=auto'
 }],

 webrtcConfig: {
  // 是否渲染多清晰度的开关，默认开启，可选
  enableAbr: true,
  // 模板名对应的label名，可选
  abrLabels: {
   d1080p: 'FHD',
   d540p: 'HD',
   d360p: 'SD',
   auto: 'AUTO',
  },
 },
});
```

这里对 WebRTC 地址中的参数做以下说明：

1. tabr_bitrates 指定了 ABR 模板，有几个模板就会渲染出几个清晰度。如果没有单独设置清晰度的 label，模板名称（如 d1080p）将被设为清晰度名称。
2. tabr_start_bitrate 指定了起播的清晰度规格。
3. tabr_control 设置是否开启自动切换清晰度。开启后，播放器会单独渲染出自动清晰度选项。

2. 手动设置清晰度

如果播放地址不是自适应码率地址，也可以手动设置清晰度。参见如下代码：

```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器ID，必须
与html中一致
```

```
  multiResolution:{
    // 配置多个清晰度地址
    sources:{
      'SD':[{
        src: 'http://video-sd-url',
      }],
      'HD':[{
        src: 'http://video-hd-url',
      }],
      'FHD':[{
        src: 'http://video-fhd-url',
      }]
    },
    // 配置每个清晰度标签
    labels:{
      'SD':'标清','HD':'高清','FHD':'超清'
    },
    // 配置各清晰度在播放器组件上的顺序
    showOrder:['SD','HD','FHD'],
    // 配置默认选中的清晰度
    defaultRes: 'SD',
  },
});
```

点播场景

在点播场景下，如果通过 fileID 播放，播放哪种规格的文件（原始文件、转码文件、自适应码率文件）以及自适应码率文件子流的清晰度，都是在播放器签名中设置的。您可以参见指引 播放自适应码流视频，以便于您了解点播场景下播放视频的整个流程。
计算播放器签名时，可以通过 contentInfo 字段中的 resolutionNames 来设定不同分辨率的子流的展示名字。不填或者填空数组则使用默认配置。

```
resolutionNames: [{
  MinEdgeLength: 240,
  Name: '240P',
}, {
  MinEdgeLength: 480,
  Name: '480P',
}, {
  MinEdgeLength: 720,
  Name: '720P',
}, {
  MinEdgeLength: 1080,
  Name: '1080P',
}, {
```

```
  MinEdgeLength: 1440,
  Name: '2K',
}, {
  MinEdgeLength: 2160,
  Name: '4K',
}, {
  MinEdgeLength: 4320,
  Name: '8K',
}]
```

播放时的子流数量取决于转码时根据不同的自适应码率模板转换出的子流数。这些子流会依据短边长度落在由 resolutionNames 设定的哪个 MinEdgeLength 范围，再以对应的 Name 作为清晰度名称进行展示。
若您需要快速体验生成播放器签名，可以使用腾讯云点播控制台的 播放器签名生成工具 。

TCPlayer 快直播降级说明

最近更新时间：2023-10-17 11:33:12

降级场景

快直播基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持。
目前，SDK 对以下操作系统和浏览器进行了测试，测试结果如下：

操作系统	操作系统版本	浏览器类型	浏览器版本	是否支持拉流
Windows	win 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		微信内嵌	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		微信内嵌	X5 内核	✓
		微信内嵌	XWeb 内核	✓

此外，在部分支持 WebRTC 的浏览器，也会出现解码失败或者服务端问题，这些情况下，播放器都会将 WebRTC 地址转换为兼容性较好的 HLS 地址来播放，这个行为称为降级处理。
总结
会触发降级的场景有以下几个：

- 浏览器环境不支持 WebRTC。

- 连接服务器失败，并且连接重试次数已超过设定值（内部状态码 -2004）。
- 播放过程解码失败（内部状态码 -2005）。
- 其他 WebRTC 相关错误（内部状态码 -2001）。

降级方式

1. 自动降级

初始化播放器时，通过 sources 字段传入了快直播地址，在需要降级处理的环境，播放器自动会进行协议的转换，将快直播地址转换为 HLS 协议地址。
例如，快直播地址：

```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=69f1eb8c
```

会自动转换为：

```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=69f1eb8c
```

2. 指定降级

在播放自适应码率（ABR）场景，如果需要降级，并不能直接通过格式转换得到自适应码率的 HLS 地址，需要手动指定。又或者是在用户希望手动指定的其他场景，都可以通过如下方式指定降级地址，这里的地址并不局限于 HLS 协议，也可以是其他协议地址。

```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=69f1eb8c&tabr_bitrates=d10
80p,d540p,d360p&tabr_start_bitrate=d1080p',
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3u8',
    },
});
```

降级回调

当触发降级时，播放器会触发回调：

```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

iOS
SDK 集成指引

最近更新时间：2024-04-26 11:00:51

本文主要介绍如何快速地将腾讯云视立方 · 播放器 LiteAVSDK_Player（iOS）集成到您的项目中，按照如下步骤进行配置，就可以完成 SDK 的集成工作。

开发环境要求

- Xcode 9.0+。
- iOS 9.0 以上的 iPhone 或者 iPad 真机。
- 项目已配置有效的开发者签名。

集成 LiteAVSDK

您可以选择使用 CocoaPods 自动加载的方式，或者先下载 SDK，再将其导入到您当前的工程项目中。

CocoaPods集成

1. **安装 CocoaPods**
   在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：

```
sudo gem install cocoapods
```

2. **创建 Podfile 文件**
   进入项目所在路径，输入以下命令行之后项目路径下会出现一个 Podfile 文件。

```
pod init
```

3. **编辑 Podfile 文件**
   使用 CocoaPod 官方源，支持选择版本号。编辑 Podfile 文件：
   Pod 方式直接集成最新版本 TXLiteAVSDK_Player：

```
platform :ios, '9.0'
source 'https://github.com/CocoaPods/Specs.git'

target 'App' do
pod 'TXLiteAVSDK_Player'
end
```

   如果您需要指定某一个特定版本，可以在 podfile 文件中添加如下依赖：

```
pod 'TXLiteAVSDK_Player', '~> 10.3.11513'
```

如果您需要集成播放高级版本（Premium）SDK，在 podfile 文件中添加如下依赖：

```
platform :ios, '9.0'
source 'https://github.com/CocoaPods/Specs.git'

target 'App' do
pod 'TXLiteAVSDK_Player_Premium'
end
```

4. **更新并安装 SDK**

- 在终端窗口中输入如下命令以更新本地库文件，并安装 LiteAVSDK：

```
pod install
```

- 或使用以下命令更新本地库版本：

```
pod update
```

pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

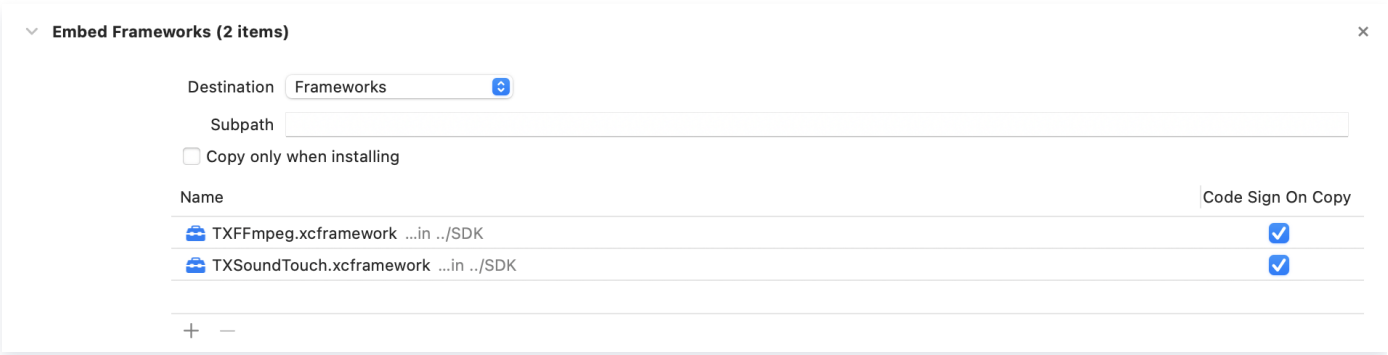
手动集成SDK

1. 下载 最新版本 TXLiteAVSDK_Player 的 SDK + Demo 开发包。
   如果您需要集成播放高级版本（Premium）SDK，请 单击此处 下载。
2. 将 SDK/TXLiteAVSDK_Player.xcframework 添加到待集成的工程中，并勾选 `Do Not Embed` 。
3. 需要配置项目 Target 的 -ObjC，否则会因为加载不到 SDK 的类别而导致 Crash。

```
打开 Xcode -> 选择对应的 Target -> 选择"Build Setting" Tab -> 搜索"Other Link Flag" ->
输入"-ObjC"
```

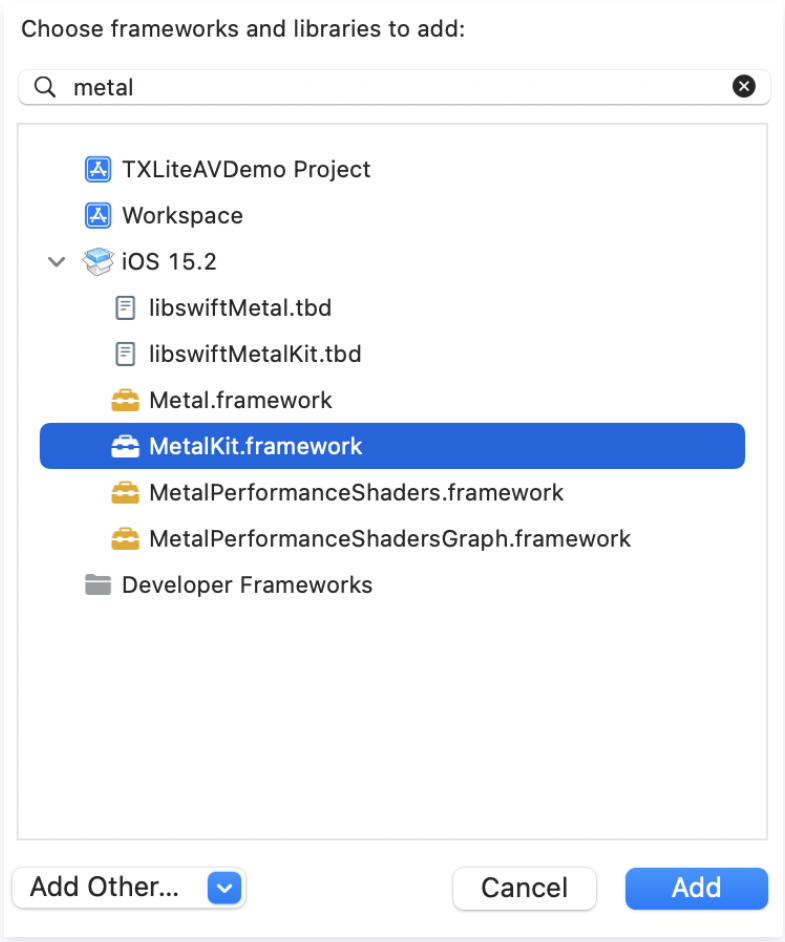
4. 添加相应的库文件（SDK 目录里）。
   - **TXFFmpeg.xcframework**：将.xcframework 文件添加到项目工程中，并在“General > Frameworks, Libraries, and Embedded Content”中将其设置为“Embed&Sign”，并在“Project Setting > Build Phases > Embed Frameworks”中进行检查，设置"Code Sign On Copy"选项为勾选状态，
   - **TXSoundTouch.xcframework**：将.xcframework 文件添加到项目工程中，并在“General > Frameworks, Libraries, and Embedded Content”中将其设置为“Embed&Sign”，并

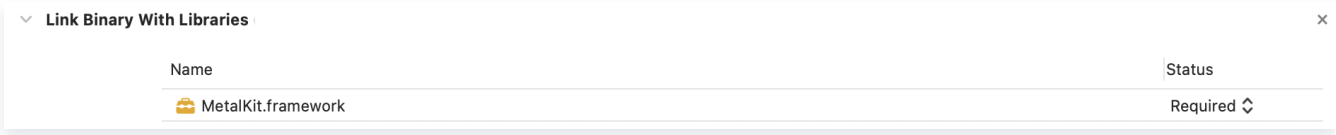
在“Project Setting > Build Phases > Embed Frameworks”中进行检查，设置 Code Sign On Copy 选项为勾选状态。



同时，切换到 Xcode 的 “Build Settings > Search Paths”，在“Framework Search Paths”中添加上述 Framework 所在的路径。

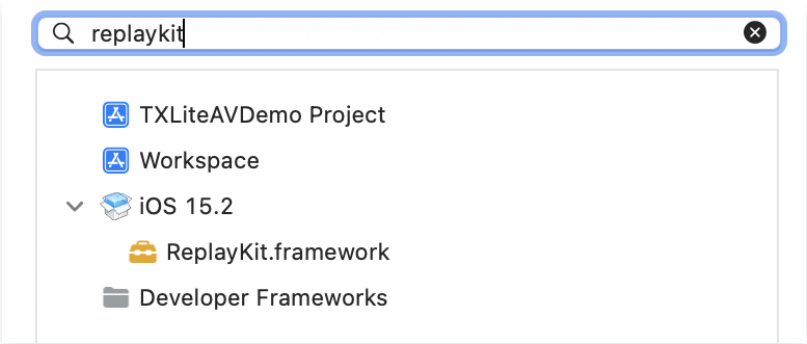
- **MetalKit.framework**：打开 Xcode，切换到“project setting > Build Phases > Link Binary With Libraries”，选择左下角的“+”号，并输入“MetalKit”，并加入项目工程中，如下图所示：

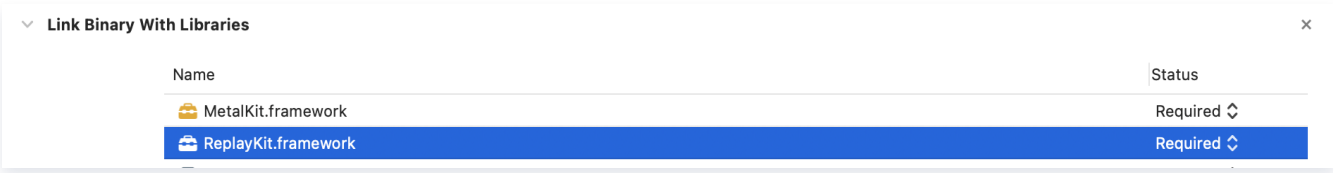




- **ReplayKit.framework**：打开 Xcode，切换到“project setting > Build Phases > Link Binary

With Libraries”，选择左下角的“+”号，并输入“ReplayKit”，并加入项目工程中，如下图所示：

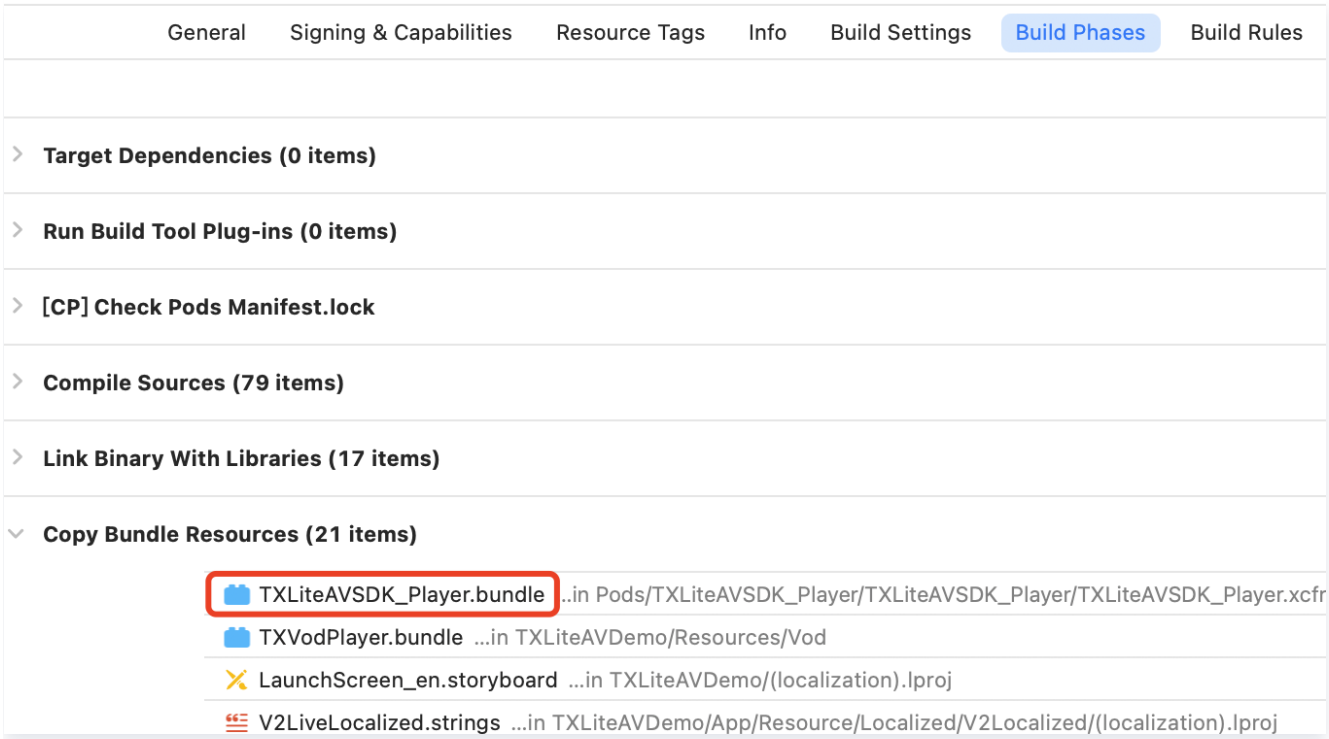




使用同样的方式添加如下系统库：

- **系统 Framework 库**：SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics, AVFoundation, Accelerate, MobileCoreServices。
- **系统 Library 库**：libz, libresolv, libiconv, libc++, libsqlite3。

5. 从 **11.7.15343** 版本 以后，Player SDK适配了苹果的隐私清单，下载对应SDK并将SDK内 **TXLiteAVSDK_Player.bundle** 添加到项目工程里：

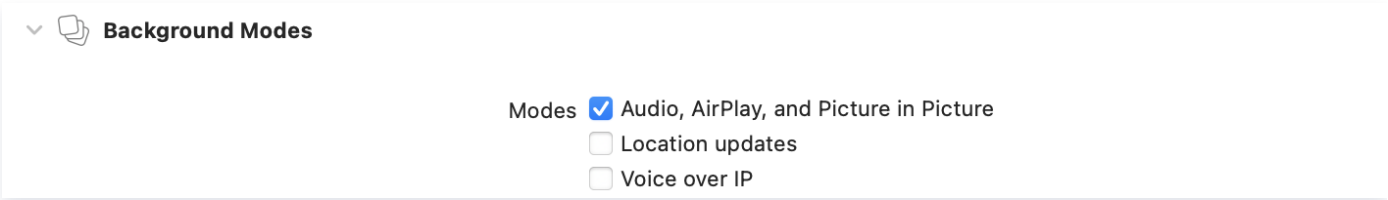


如果是 pods 引入 SDK 的话，可以忽略上述步骤。

画中画功能

如果需要使用画中画能力，请按如下图的方式进行配置，若无此部分需求可以忽略。

1. 为了使用 iOS 的画中画（Picture-In-Picture），请将 SDK 升级到10.3版本及以上。
2. 使用画中画能力时，需要开通后台模式。XCode 选择对应的Target > Signing & Capabilities > Background Modes，勾选“Audio, AirPlay, and Picture in Picture”，如图所示：



在工程中引入 SDK

项目代码中使用 SDK 有两种方式：

- **方式一：** 在项目需要使用 SDK API 的文件里，添加模块引用。

```
@import TXLiteAVSDK_Player;
// 如果您使用的的 Premium 版本，请用： @import TXLiteAVSDK_Player_Premium;
```

- **方式二：**在项目需要使用 SDK API 的文件里，引入具体的头文件。

```
#import "TXLiteAVSDK_Player/TXLiteAVSDK.h"
// 如果您使用的的 Premium 版本，请用： #import "TXLiteAVSDK_Player_Premium/TXLiteAVSDK.h"
```

给 SDK 配置 License 授权

1. 单击 License 申请 获取测试用 License，不配置 License 将会播放时视频失败，具体操作请参见 测试版 License 。您会获得两个字符串：一个字符串是 licenseURL，另一个字符串是解密 key。
2. 获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License 。

常见问题

项目里面同时集成了直播 SDK/实时音视频/播放器等 LiteAVSDK 系列的多个 SDK 报符号冲突问题怎么解决?

如果集成了2个或以上产品（直播、播放器、TRTC、短视频）的 LiteAVSDK 版本，编译时会出现库冲突问题，因为有些 SDK 底层库有相同符号文件，这里建议只集成一个全功能版 SDK 可以解决，直播、播放器、TRTC、短视频这些都包含在一个 SDK 里面。具体请参见 SDK 下载 。

Swift 项目工程里怎么调用 SDK 的 API 方法?

如果在 Swift 的项目工程里想调用 SDK 的 API 接口，有下面两种方式：

方式一：使用桥接头文件

1. 创建桥接头文件。例如 `***-Bridging-Header.h` ，并添加如下代码 `#import <TXLiteAVSDK_Player/TXLiteAVSDK.h>` 。
2. 配置工程 `BuildSetting` 的 `Objective-c Bridging header` 选项。设置桥接文件的路径并添加到 `Objective-c Bridging header` 中（ 如：$(SRCROOT)/SwiftCallOC/***-Bridging-Header.h，根据项目具体路径确定），编译运行即可。

方式二：使用SDK内的 module.modulemap 文件

1. 检查 `TXLiteAVSDK_Player.framework` 里是否有包含 `Modules - module.modulemap` 文件（Player SDK 默认都提供）。
2. 配置工程 `BuildSetting` 的 `Swift Compiler - Search Paths` 选项。添加 `module.modulemap` 文件所在的目录路径或其上层目录路径，此处可为：${PODS_ROOT}/TXLiteAVSDK_Player/TXLiteAVSDK_Player/TXLiteAVSDK_Player.framework/Modules（根据项目具体路径确定）。
3. 在需要调用的类顶部，使用 `import TXLiteAVSDK_Player` 来进行引入并调用相关的方法。

以上集成的方式及 Demo，可以具体参见 GitHub Demo 。

点播场景

最近更新时间：2024-05-07 14:22:21

准备工作

1. 开通 云点播 相关服务，未注册用户可注册账号 试用 。
2. 下载 Xcode，如您已下载可略过该步骤，您可以进入 App Store 下载安装。
3. 下载 Cocoapods，如您已下载可略过该步骤，您可以进入 Cocoapods 官网 按照指引进行安装。

通过本文您可以学会

- 如何集成腾讯云视立方 iOS 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。
- 播放器推出短视频组件、画中画2.0、VR 播放等高级组件，功能介绍和使用指引请参见 移动端高级功能 。

SDK 集成

步骤1：集成 SDK 开发包

下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引 。

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 播放器 License 获取相关授权。
- 获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License 。

步骤3：创建 Player

视频云 SDK 中的 TXVodPlayer 模块负责实现点播播放功能。

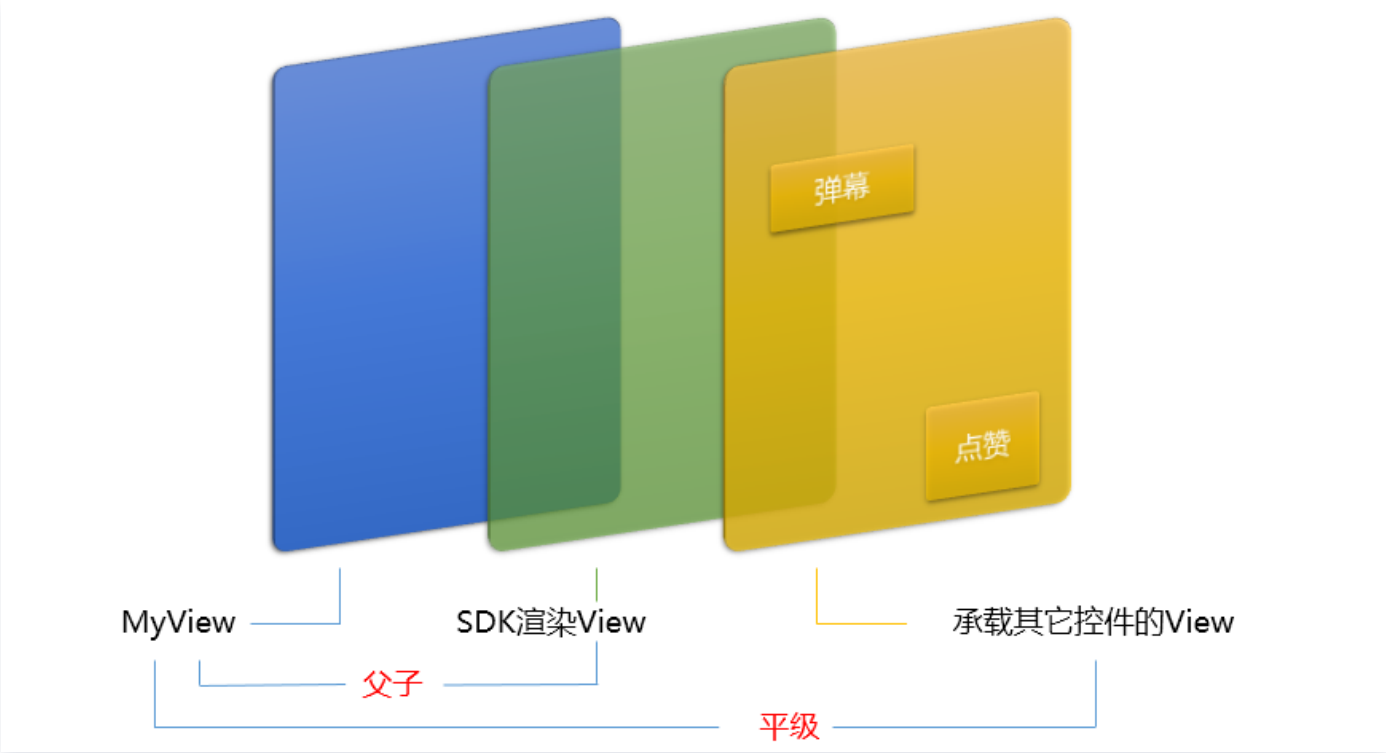
```
TXVodPlayer *_txVodPlayer = [[TXVodPlayer alloc] init];
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

步骤4：渲染 View

接下来我们要给播放器的视频画面找个地方来显示，iOS 系统中使用 view 作为基本的界面渲染单位，所以您只需要准备一个 view 并调整好布局就可以了。

```
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

内部原理上讲，播放器并不是直接把画面渲染到您提供的 view（示例代码中的 _myView）上，而是在这个 view 之上创建一个用于 OpenGL 渲染的子视图（subView）。
如果您要调整渲染画面的大小，只需要调整您所常见的 view 的大小和位置即可，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。



如何做动画

针对 view 做动画是比较自由的，不过请注意此处动画所修改的目标属性应该是 transform 属性而不是 frame 属性。

```
[UIView animateWithDuration:0.5 animations:^{
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); // 缩小1/3
 }];
```

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
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startVodPlay:url];

// 播放沙盒本地视频资源
// 获取 Documents路径
NSString *documentPath =
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
YES) firstObject];
// 获取本地视频路径
NSString *videoPath = [NSString
stringWithFormat:@"%@/video1.m3u8",documentPath];
[_txVodPlayer startVodPlay:videoPath];
```

通过 fileId 方式

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788;
p.fileId = @"4564972819220421305";
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
p.sign = @"psignxxxxx"; // 播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

在 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 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 `PLAY_ERR_GET_PLAYINFO_FAIL` 事件，反之收到 `PLAY_EVT_GET_PLAYINFO_SUCC` 表示请求成功。

步骤6：结束播放

结束播放时，如果要退出当前的 UI 界面，要记得用 **removeVideoWidget** 销毁 view 控件，否则会产生内存泄露或闪屏问题。

```
// 停止播放
[_txVodPlayer stopPlay];
[_txVodPlayer removeVideoWidget]; // 记得销毁 view 控件
```

基础功能使用

1、播放控制

开始播放

```
// 开始播放
[_txVodPlayer startVodPlay:url];
```

暂停播放

```
// 暂停播放
[_txVodPlayer pause];
```

恢复播放

```
// 恢复播放
[_txVodPlayer resume];
```

结束播放

```
// 结束播放
[_txVodPlayer stopPlay];
```

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 默认支持精准 seek，可以在播放器前通过 `TXVodPlayConfig#setEnableAccurateSeek` 进行配置。

```
float time = 600; // int 类型时，单位为 秒
// 调整进度
[_txVodPlayer seek:time];
```

精准和非精准 Seek

播放器 SDK 11.8 版本开始，支持调用 seek 接口时，指定精准或非精准 seek。

```
float time = 600; // float 类型时单位为 秒
// 调整进度
[_txVodPlayer seek:time accurateSeek:YES];  // 精准 seek
[_txVodPlayer seek:time accurateSeek:NO];  // 非精准 seek
```

Seek 到视频流指定 PDT 时间点

跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。
注意：播放器高级版 11.6 版本开始支持。

```
long long pdtTimeMs = 600; // 单位为 毫秒
[_txVodPlayer seekToPdtTime:time];
```

从指定时间开始播放

首次调用 startVodPlay之前，支持从指定时间开始播放。

```
float startTimeInSecond = 60; // 单位：秒
[_txVodPlayer setStartTime:startTimeInSecond];  // 设置开始播放时间
[_txVodPlayer startVodPlay: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 setupVideoWidget 的参数 view 的大小和位置，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。
- **setRenderMode：铺满或适应**

可选值	含义
RENDER_MODE_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_FILL_EDGE	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
HOME_ORIENTATION_RIGHT	home 在右边

HOME_ORIENTATION_DOWN	home 在下面
HOME_ORIENTATION_LEFT	home 在左边
HOME_ORIENTATION_UP	home 在上面

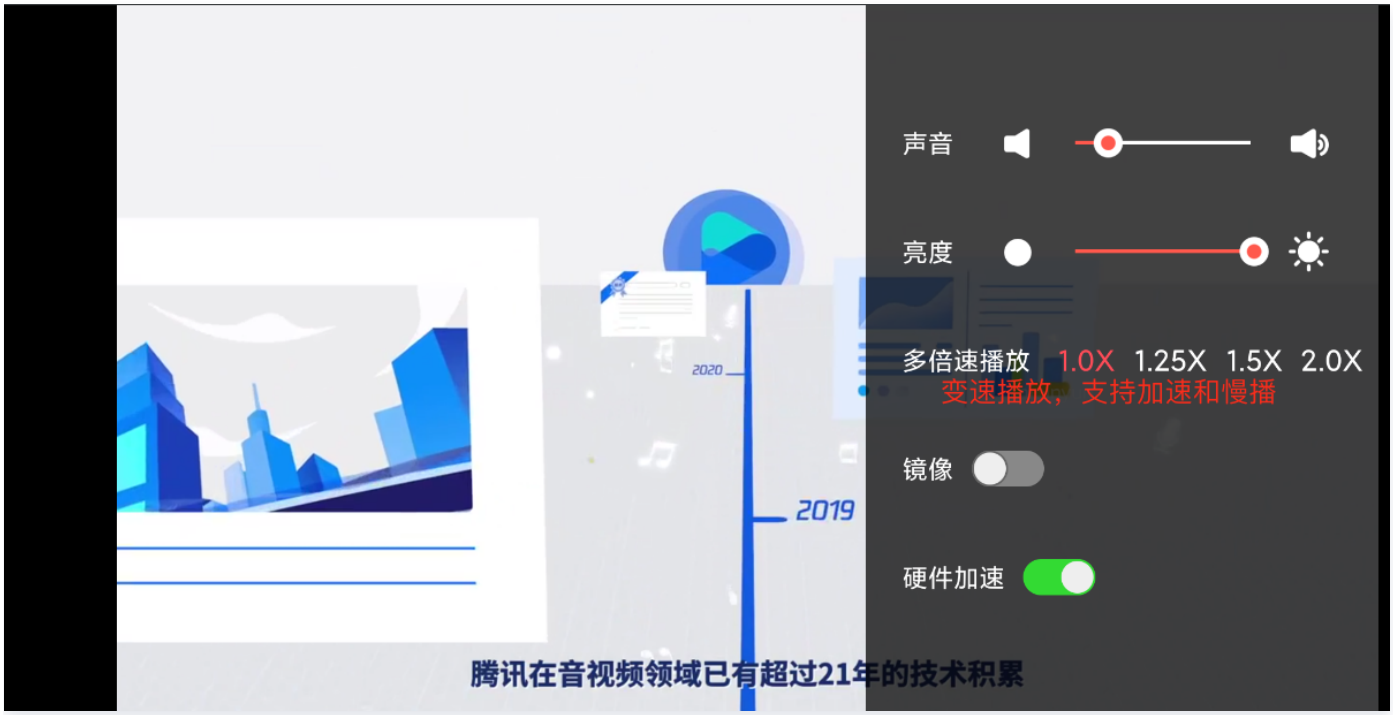
最长边填充

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
[_txVodPlayer setRate:1.2];
// 开始播放
[_txVodPlayer startVodPlay:url];
```

4、循环播放

```
// 设置循环播放
[_txVodPlayer setLoop:true];
// 获取当前循环播放状态
[_txVodPlayer loop];
```

5、静音设置

```
// 设置静音，true 表示开启静音， false 表示关闭静音
[_txVodPlayer 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 iOS 的系统 API 来实现。

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。

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。

```
NSMutableDictionary<NSString *, NSString *> *httpHeader = [[NSMutableDictionary
alloc] init];
[httpHeader setObject:@"${Referer Content}" forKey:@"Referer"];
[_config setHeaders:httpHeader];
[_txVodPlayer setConfig:_config];
```

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 **startVodPlay**，否则会产生比较严重的花屏问题。

```
[_txVodPlayer stopPlay];
_txVodPlayer.enableHWAcceleration = YES;
[_txVodPlayer startVodPlay:_flvUrl type:_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。可以通过下面方法获取多码率数组。

```
// 在收到播放器 PLAY_EVT_VOD_PLAY_PREPARED 事件调用 getSupportedBitrates 才会有值返回
NSArray *bitrates = [_txVodPlayer supportedBitrates]; //获取多码率数组
// TXBitrateItem 类字段含义：index-码率下标；width-视频宽；height-视频高；birate-视频码率
TXBitrateItem *item = [bitrates objectAtIndex:i];
[_txVodPlayer setBitrateIndex:item.index]; // 切换码率到想要的清晰度

// 获取当前播放的码率下标，返回值 -1000 为默认值，表示没有设置过码率标；返回值 -1 表示开启了自适应码流
int index = [_txVodPlayer bitrateIndex];
```

在播放过程中，可以随时通过 `-[TXVodPlayer setBitrateIndex:]` 切换码率。切换过程中，会重新拉取另一条流的数据，因此会有稍许卡顿。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。
如果您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，从而避免播放后切换码流。详细方法参考 播放器配置#启播前指定分辨率 。

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。可以通过下面方法开启码流自适应：

```
[_txVodPlayer setBitrateIndex:-1]; //index 参数传入-1
```

在播放过程中，可以随时通过 `-[TXVodPlayer setBitrateIndex:]` 切换其它码率，切换后码流自适应也随之关闭。

12、开启平滑切换码率

在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
// 设为 YES，在 IDR 对齐时可平滑切换码率，设为 NO 时，可提高多码率地址打开速度
[_config setSmoothSwitchBitrate:YES];
[_txVodPlayer setConfig:_config];
```

13、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 。


```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:
(NSDictionary*)param {
    if (EvtID == PLAY_EVT_PLAY_PROGRESS) {
        // 加载进度，单位是秒，小数部分为毫秒
        float playable = [param[EVT_PLAYABLE_DURATION] floatValue];
            [_loadProgressBar setValue:playable];

        // 播放进度，单位是秒，小数部分为毫秒
        float progress = [param[EVT_PLAY_PROGRESS] floatValue];
            [_seekProgressBar setValue:progress];

        // 视频总长，单位是秒，小数部分为毫秒
        float duration = [param[EVT_PLAY_DURATION] floatValue];
        // 可以用于设置时长显示等等

         // 获取 PDT 时间，播放器高级版 11.6 版本开始支持
        long long pdt_time_ms = [param[VOD_PLAY_EVENT_PLAY_PDT_TIME_MS]
longLongValue];
    }
}
```

14、播放网速监听

通过 事件监听 方式，可以在视频播放卡顿时显示当前网速。

- **通过** `onNetStatus` **的** `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。

15、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 。

分辨率信息

- **方法1：** 通过 onNetStatus 的 VIDEO_WIDTH 和 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：** 直接调用 -[TXVodPlayer width] 和 -[TXVodPlayer height] 获取当前宽高。

16、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 = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10];  // 播放时最大缓冲大小。单位：MB
[_txVodPlayer setConfig:_config];  // 把 config 传给 _txVodPlayer
```

17、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。
- **开启方法：** 开启此功能需要配置两个参数：本地缓存目录及缓存大小。

```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory
stringByAppendingPathComponent:@"/preload"];
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
   [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                          withIntermediateDirectories:NO
                                           attributes:nil
                                                error:&error];
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];
// 开始播放
[_txVodPlayer startVodPlay:url];
```

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

18、屏幕控制(亮屏和灭屏)

由于手机个性化设置中经常会设置屏幕锁定的时间，在视频播放场景中可能会出现屏幕灭屏（或被锁定）的情况，这极大影响了用户体验。因此，为了解决这一情况需要在播放过程中的相关时机添加如下代码，以便使屏幕一直处于亮屏状态。

```
（1）亮屏(禁止灭屏)
// 启动播放（startVodPlay / startPlayDrm / startVodPlayWithParams）
// 恢复播放（resume）
[[UIApplication sharedApplication] setIdleTimerDisabled:YES];

（2）灭屏（恢复灭屏）
// 停止（stopPlay）
// 暂停（pause）
[[UIApplication sharedApplication] setIdleTimerDisabled:NO];
```

> **注意：**
> 以上接口的使用请注意在 主线程 调用。

19、DRM 加密视频播放

> **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持播放商业级 DRM 加密视频，目前支持 WideVine 和 Fairplay 两种 DRM 方案。更多的商业级 DRM 信息，请参考 商业级 DRM 综述 。
可通过下面两种方式播放 DRM 加密视频：

通过 FileId 播放

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = ${appId}; // 腾讯云账户的 appId
p.fileId = @"${fileId}"; // DRM 加密视频的 fileId
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
p.sign = @"${psgin}"; // 加密视频的播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

通过 FileId 播放适用于接入云点播后台。这种方式和播放普通 FileId 文件没有差别，需要在云点播先配置资源为 DRM 类型，SDK 会在内部识别并处理。

自定义配置播放

```
// 通过 TXVodPlayer#startPlayDrm 接口播放
// @param certificateUrl 证书提供商url
// @param licenseUrl 解密的key url
// @param videoUrl 待播放视频的Url地址
TXPlayerDrmBuilder *builder = [[TXPlayerDrmBuilder alloc]
initWithDeviceCertificateUrl:@"${certificateUrl}" licenseUrl:@"${licenseUrl}"
videoUrl:@"${videoUrl}"];
[_txVodPlayer startPlayDrm:builder];
```

20、外挂字幕

> ⚠ **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这两种格式的字幕。
最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。添加字幕后，并不会主动加载字幕，调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。
用法如下：
步骤1：添加外挂字幕。

```
// 传入 字幕url， 字幕名称， 字幕类型，建议在启动播放器前添加
[_txVodPlayer addSubtitleSource:@"https://mediacloud-76607.gzc.vod.tencent-
cloud.com/DemoResource/subtitleVTT.vtt" name:@"subtitleName"
mimeType:TX_VOD_PLAYER_MIMETYPE_TEXT_VTT];
```

步骤2：播放后切换字幕。

```
// 开始播放视频后，选中添加的外挂字幕，请在收到 VOD_PLAY_EVT_SELECT_TRACK_COMPLETE
事件后调用
NSArray<TXTrackInfo *> *subtitlesArray = [_txVodPlayer getSubtitleTrackInfo];
for (int i = 0; i < subtitlesArray.count; i++) {
    TXTrackInfo *info = subtitlesArray[i];
```

```
    if (info.trackIndex == 0) {
        [_txVodPlayer selectTrack:info.trackIndex];  // 选中字幕
    } else {
        // 其它字幕不需要的话， 进行deselectTrack
        [_txVodPlayer deselectTrack:info.trackIndex];
    }
}

// 监听轨道切换消息
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
*)param {
    if (EvtID == VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
        int trackIndex = [(NSNumber *)[param valueForKey:EVT_KEY_SELECT_TRACK_INDEX]
intValue];
        int errorCode = [(NSNumber *)[param
valueForKey:EVT_KEY_SELECT_TRACK_ERROR_CODE] intValue];
        NSLog(@"receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=%d ,
errorCode=%d", trackIndex, errorCode);
    }
}
```

步骤3：配置字幕样式。

字幕样式支持在播放前或者播放过程中配置。

```
// 详细参数配置请参考 API 文档
TXPlayerSubtitleRenderModel *model = [[TXPlayerSubtitleRenderModel alloc] init];
model.canvasWidth = 1920;  // 字幕渲染画布的宽
model.canvasHeight = 1080;  // 字幕渲染画布的高
model.isBondFontStyle = NO;  // 设置字幕字体是否为粗体
model.fontColor = 0xFF000000; // 设置字幕字体颜色，默认白色不透明
[_txVodPlayer setSubtitleStyle:model];
```

21、多音轨切换

> **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持切换视频内置的多音轨。用法如下：

```
NSArray<TXTrackInfo *> *soundTrackArray = [_txVodPlayer getAudioTrackInfo];
for (int i = 0; i < soundTrackArray.count; i++) {
    TXTrackInfo *info = soundTrackArray[i];
    if (info.trackIndex == 0) {
        // 通过判断 trackIndex 或者 name 切换到需要的音轨
```

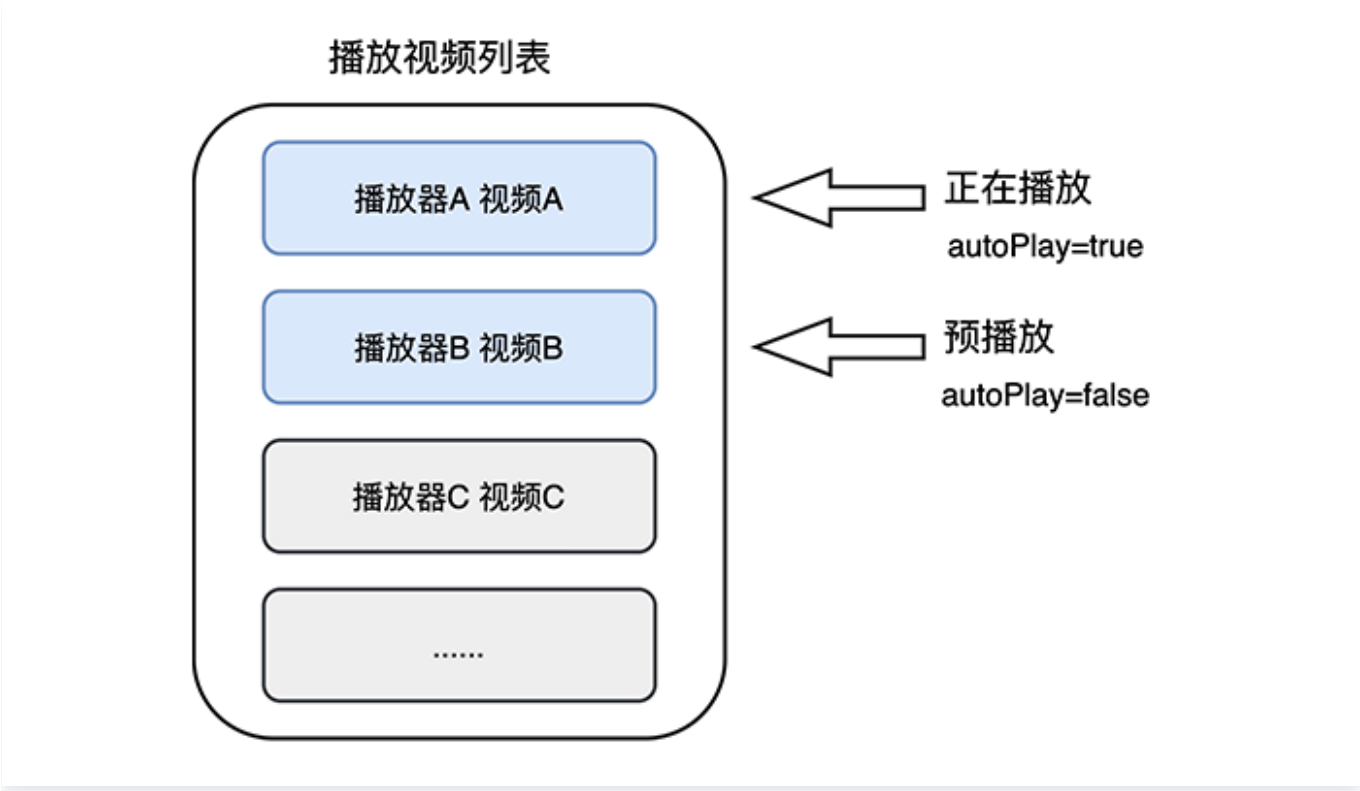
```
        [_txVodPlayer selectTrack:info.trackIndex];
    } else {
        // 其它字幕不需要的话， 进行 deselectTrack
        [_txVodPlayer deselectTrack:info.trackIndex];
    }
}
```

进阶功能使用

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，预加载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 URL，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 isAutoPlay 开关来实现这个功能，具体做法如下：




```
// 播放视频 A: 如果将 isAutoPlay 设置为 YES， 那么 startVodPlay调用会立刻开始视频的加载和播放
NSString* url_A = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
_player_A.isAutoPlay = YES;
[_player_A startVodPlay:url_A];
```

```
// 在播放视频 A 的同时，预加载视频 B，做法是将 isAutoPlay 设置为 NO
NSString* url_B = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
_player_B.isAutoPlay = NO;
[_player_B startVodPlay:url_B];
```

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。

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:
(NSDictionary*)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 (EvtID == PLAY_EVT_PLAY_END) {
            [_player_A stopPlay];
            [_player_B setupVideoWidget:mVideoContainer insertIndex:0];
            [_player_B 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 = [[TXVodPlayConfig alloc]init];
[_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
[_txVodPlayer setConfig:_config]; // 把config 传给 _txVodPlayer
```

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 = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10]; // 播放时最大缓冲大小。单位：MB
```

```
[_txVodPlayer setConfig:_config];  // 把 config 传给 _txVodPlayer
```

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：**
> - TXPlayerGlobalSetting 是全局缓存设置接口，原有 TXVodConfig 的缓存配置接口废弃。
> - 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。

通过媒资 URL 预下载

通过媒资 URL 预下载视频代码示例如下：

```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory
stringByAppendingPathComponent:@"/preload"];
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                                  withIntermediateDirectories:NO
                                                   attributes:nil
                                                        error:&error]; //Create
folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];

//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];
NSString *m3u8url = "http://****";
int taskID = [[TXVodPreloadManager sharedManager] startPreload:m3u8url
                                                    preloadSize:10

preferredResolution:1920*1080
                                                       delegate:self];


//取消预下载
[[TXVodPreloadManager sharedManager] stopPreload:taskID];
```

```
- (void)onComplete:(int)taskID
              url:(NSString *)url {
    /**
     * 下载完成回调
     * taskID 下载任务ID
     * url 下载任务地址
     */
}
- (void)onError:(int)taskID
            url:(NSString *)url
          error:(NSError *)error {
    /**
     * 下载错误回调
     * taskID 下载任务ID
     * url 下载任务地址
     * error 下载失败的错误信息
     */
}
```

通过媒资 FileId 预下载

> **注意：**
> 通过 fileId 预下载从 11.3 版本开始支持。

通过 fileId 预下载是耗时操作， 请不要在主线程调用，否则会抛出非法调用异常。startPreload 时传入的 preferredResolution 要和启播时设置的优先启播分辨率保持一致，否则将达不到预期的效果。 `onStart` 回调的 URL 可以保存起来，传给播放器播放。使用示例如下：

```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory
stringByAppendingPathComponent:@"/preload"];
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                              withIntermediateDirectories:NO
                                               attributes:nil
                                                    error:&error]; //Create folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
```

```
//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];

TXPlayerAuthParams *params = [[TXPlayerAuthParams alloc] init];
params.appId = ${appId};
params.fileId = @"${fileId}";
params.sign = @"${psign}";
// 注意： 耗时操作，请不要在主线程调用！在主线程调用将会抛出非法调用异常。
int taskID = [[TXVodPreloadManager sharedManager] startPreload:params
                                                   preloadSize:10
                                            preferredResolution:1920*1080
                                                      delegate:self]; // TXVodPreloadManagerDelegate

//取消预下载
[[TXVodPreloadManager sharedManager] stopPreload:taskID];

//预下载TXVodPreloadManagerDelegate代理方法实现
- (void)onStart:(int)taskID
         fileId:(NSString *)fileId
            url:(NSString *)url
          param:(NSDictionary *)param {
          /**
           * 启动下载（此方法在换链成功以后，启动下载之前回调）
           * taskID 下载任务ID
           * fileId 下载视频的 fileId。URL方式缓存时，此参数为nil
           * url 下载任务地址
           * param 附加参数
           */
}
- (void)onComplete:(int)taskID
               url:(NSString *)url {
           /**
            * 下载完成回调
            * taskID 下载任务ID
            * url 下载任务地址
            */
}
- (void)onError:(int)taskID
            url:(NSString *)url
          error:(NSError *)error {
          /**
           * 下载错误回调
           * taskID 下载任务ID
           * url 下载任务地址
           * error 下载失败的错误信息
           */
}
```

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。如果是加密视频，通过播放器 SDK 下载后的视频在本地保持为加密状态，仅可通过腾讯云播放器 SDK 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 离线播放，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 。
> - 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

SDK 初始化时，设置全局存储路径，用于视频下载，预加载，和缓存等功能。用法如下：

```
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES)            firstObject];
NSString downloadPath = [NSString stringWithFormat:@"%@/txdownload",cachesDir];
[TXPlayerGlobalSetting setCacheFolderPath:downloadPath];
```

`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 shareInstance];
```

步骤2：开始下载

开始下载有两种方式：fileid 和 URL。

fileid 方式

fileid 下载至少需要传入 appId 和 fileId，userName 不传入具体值时，默认为 “default”。**注意：加密视频只能通过 Fileid 下载。**

```
TXVodDownloadDataSource *source = [[TXVodDownloadDataSource alloc] init];
source.appId = 1252463788;
source.fileId = @"4564972819220421305";
// // psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
```

```
source.pSign = @"xxxxxxxxxx";

// 指定下载清晰度
// 可用枚举值有：240p:TXVodQuality240P, 360p:TXVodQuality360P,
480p:TXVodQuality480P, 540p:TXVodQuality540P, 720p:TXVodQuality720P,
// 1080p:TXVodQuality1080P,2K:TXVodQuality2K,4K:TXVodQuality4K
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720, 期望下载此分辨率的
流，quality传入 TXVodQuality720P)
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
source.quality = TXVodQuality720P;  // 720p

// ** 注意如果是使用旧的 v2 协议下载，请通过 TXVodDownloadDataSource中的 auth 属性
来设置 appId 和 fileId 这些参数 **
// source.auth = auth;  ** 默认无需设置 **

[downloader startDownload:dataSource];
```

URL 方式

至少需要传入下载地址 URL。preferredResolution 取值为视频分辨率宽和高的乘积：preferredResolution=width * height。如果是嵌套 HLS 格式，preferredResolution 不传入具体值时，默认值为921600。userName 不传入具体值时，默认为"default"。私有加密请使用 fileid 形式。

```
[downloader startDownloadUrl:@"" resolution:@921600 userName:@""];
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 delegate。

```
downloader.delegate = self;
```

可能收到的任务回调有：

回调信息	含义
-[TXVodDownloadDelegate onDownloadStart:]	任务开始，表示 SDK 已经开始下载。
-[TXVodDownloadDelegate onDownloadProgress:]	任务进度，下载过程中，SDK 会频繁回调此接口，您可以在这里更新进度显示。

-[TXVodDownloadDelegate onDownloadStop:]	任务停止，当您调用是 stopDownload 停止下载，收到此消息表示停止成功。
-[TXVodDownloadDelegate onDownloadFinish:]	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。
-[TXVodDownloadDelegate onDownloadError:errorMsg:]	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。所有错误码请参考 TXDownloadError。

下载错误码

错误码	数值	含义说明
TXDownloadSuccess	0	下载成功。
TXDownloadAuthFaild	-5001	向云点播控制台请求视频信息失败，建议检查fileId、psign参数是否正确。
TXDownloadNoFile	-5003	无此清晰度文件。
TXDownloadFormatError	-5004	下载文件格式不支持。
TXDownloadDisconnet	-5005	网络断开，建议检查网络是否正常。
TXDownloadHlsKeyError	-5006	获取 HLS 解密 Key 失败。
TXDownloadPathError	-5007	下载目录访问失败，建议检查是否有访问下载目录的权限。

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 `-[TXVodDownloadManager stopDownload:]` 方法，参数为 `-[TXVodDownloadManager sartDownloadUrl:]` 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

1. 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。

```
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
NSArray<TXVodDownloadMediaInfo *> *array = [[[TXVodDownloadManager
shareInstance] getDownloadMediaInfoList] mutableCopy];
// 获取默认 “default” 用户的下载列表
```

```
for (TXVodDownloadMediaInfo *info in array) {
    if ([info.userName isEqualToString:@"default"]) {
        // 保存 “default” 用户的下载列表
    }
}
```

2. 获取 FileId 或 URL 相关的下载信息：

2.1 通过接口 `-[TXVodDownloadManager getDownloadMediaInfo:]` 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 *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc]
init];
TXVodDownloadDataSource *dataSource = [[TXVodDownloadDataSource alloc]
init];
dataSource.appId = 1252463788;
dataSource.fileId = @"4564972819220421305";
dataSource.pSign = @"psignxxxx";
dataSource.quality = TXVodQualityHD;
sourceMediaInfo.dataSource = dataSource;
// getDownloadMediaInfo 是耗时接口，请不要在主线程调用
TXVodDownloadMediaInfo *downlaodMediaInfo = [[TXVodDownloadManager
shareInstance] getDownloadMediaInfo:sourceMediaInfo];

// 获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。
// 备注：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大
小，暂时无法获取。
downlaodMediaInfo.size; // 获取下载文件总大小
downlaodMediaInfo.duration;  // 获取总时长
downlaodMediaInfo.playableDuration; // 获取已下载的可播放时长
downlaodMediaInfo.progress;  // 获取下载进度
downlaodMediaInfo.playPath;  // 获取离线播放路径，传给播放器即可离线播放
downlaodMediaInfo.downloadState;  // 获取下载状态，具体参考 STATE_xxx 常量
[downlaodMediaInfo isDownloadFinished];  // 返回 YES 表示下载完成
```

2.2 获取某个 URL 相关下载信息，需要传入 URL 信息即可。

```
// 获取某个 fileId 相关下载信息
TXVodDownloadMediaInfo *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc]
init];
mediaInfo.url = @"videoURL";
TXVodDownloadMediaInfo *downlaodMediaInfo = [[TXVodDownloadManager
shareInstance] getDownloadMediaInfo:sourceMediaInfo];
```

3. 删除下载信息和相关文件：

如果您不需要重新下载，请调用 `-[TXVodDownloadManager deleteDownloadFile:]` 方法删除文件，以释放存储空间。

步骤6：下载后离线播放

下载后的视频支持无网络的情况下进行播放，无需进行联网。下载完成后，即可进行播放。

```
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
NSArray<TXVodDownloadMediaInfo *> *mediaInfoList = [[TXVodDownloadManager
shareInstance] getDownloadMediaInfoList];
TXVodDownloadMediaInfo *mediaInfo = [mediaInfoList firstObject]; // 根据情况找到当前的
media 对象
if (mediaInfo.downloadState == TXVodDownloadMediaInfoStateFinish) { // 判断是否下载完
成
    [self.player startVodPlay:mediaInfo.playPath];
}
```

> **注意：**
> 离线下载播放时，一定要通过获取下载列表并通过下载列表视频对象 TXVodDownloadMediaInfo 的 `PlayPath` 进行播放，切勿直接保存 PlayPath 对象。

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。
在腾讯云控制台提取到 appId， 加密视频的 fileId 和 psign 后，可以通过下面的方式进行播放：

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788; // 腾讯云账户的 appId
p.fileId = @"4564972819220421305"; // 视频的 fileId
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
p.sign = @"psignxxxxx"; // 播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

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，比如：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 = [[TXVodPlayConfig alloc]init];
```

```
[_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
[_txVodPlayer setConfig:_config]; // 把config 传给 _txVodPlayer
```

- **启播前指定分辨率**

  播放 HLS 的多码率视频源，如果您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 = [[TXVodPlayConfig alloc]init];
// 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:720*1280];
[_txVodPlayer setConfig:_config]; // 把config 传给 _txVodPlayer
```

- **启播前指定媒资类型**

当提前知道播放的媒资类型时，可以通过配置 `TXVodPlayConfig#setMediaType` 减少播放器SDK内部播放类型探测，提升启播速度。

> **注意：**
> `TXVodPlayConfig#setMediaType` 11.2 版本开始支持。

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMediaType:MEDIA_TYPE_FILE_VOD];  // 用于提升MP4启播速度
// [_config setMediaType:MEDIA_TYPE_HLS_VOD];  // 用于提升HLS启播速度
[_txVodPlayer setConfig:_config];
```

- **设置播放进度回调时间间隔**

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setProgressInterval:200]; // 设置进度回调间隔，单位毫秒
[_txVodPlayer setConfig:_config]; // 把 config 传给 _txVodPlayer
```

6、HttpDNS 解析服务

移动解析（HTTPDNS）基于 HTTP 协议向 DNS 服务器发送域名解析请求，替代了基于 DNS 协议向运营商 Local DNS 发起解析请求的传统方式，可避免 Local DNS 造成域名劫持和跨网访问问题，解决移动互联网服务中域名解析异常带来的视频播放失败困扰。

> **注意：**
> HttpDNS 解析服务从 10.9 版本开始支持。

1. 开通 HTTPDNS 解析服务。

   您可以选择腾讯云或其它云提供商，开通 HTTPDNS 解析服务，确保开通成功后， 再集成到播放 SDK。
2. 在播放 SDK 接入 HTTPDNS 解析服务。

   下面以接入 腾讯云 HTTPDNS 为例子，展示如何在播放器 SDK 接入：

```
// 步骤1： 打开 HttpDNS 解析开关
[TXLiveBase enableCustomHttpDNS:YES];
// 步骤2： 实现 HttpDNS 解析代理：TXLiveBaseDelegate#onCustomHttpDNS
- (void)onCustomHttpDNS:(NSString *)hostName ipList:(NSMutableArray<NSString *>
*)list {
    // 把 hostName 解析到 ip 地址后，保存到 iPList，返回给 SDK 内部。注意：这里不要进行耗
时的异步操作。
    // MSDKDnsResolver 是腾讯云提供的 HTTPDNS SDK 解析接口
    NSArray *result = [[MSDKDns sharedInstance] WGGetHostByName:hostName];
    NSString *ip = nil;
    if (result && result.count > 1) {
        if (![result[1] isEqualToString:@"0"]) {
            ip = result[1];
        } else {
            ip = result[0];
        }
    }
    [list addObject:ip];
}

// 步骤3： 设置 HttpDNS 解析代理
[TXLiveBase sharedInstance].delegate = self;
```

7、HEVC 自适应降级播放

播放器支持同时传入 HEVC 和其它视频编码格式比如： H.264 的播放链接， 当播放机型不支持 HEVC 格式时，将自动降级为配置的其它编码格式（如： H.264 ）的视频播放。
注意：播放器高级版 11.7 版本开始支持。

```
#import <CoreMedia/CoreMedia.h> // 引入头文件

NSDictionary *dic = @{
    VOD_KEY_VIDEO_CODEC_TYPE:@(kCMVideoCodecType_HEVC), // 指定原始 HEVC 视频编
码类型
    VOD_KEY_BACKUP_URL:@"${backupPlayUrl}"}; // 设置 H.264 格式等备选播放链接地址
[_txVodPlayer setExtentOptionInfo:dic];

// 设置原始 HEVC 播放链接
[_txVodPlayer startVodPlay:@"${hevcPlayUrl}"];
```

8、音量均衡

播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。通过 `TXVodPlayer#setAudioNormalization` 设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。
注意：播放器高级版 11.7 版本开始支持。

```
/**
 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h）
 关：AUDIO_NORMALIZATION_OFF
 开：AUDIO_NORMALIZATION_STANDARD（标准）
   AUDIO_NORMALIZATION_LOW （低）
   AUDIO_NORMALIZATION_HIGH（高）
 可填自定义数值：从低到高，范围-70 - 0 LUFS
*/
[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_STANDARD];     //启动音量均衡

[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_OFF];          //关闭音量均衡
```

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。

事件通知（onPlayEvent）

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_END	2006	视频播放结束
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_EN	201	视频播放 loading 结束，视频继续播放。

D	4	
VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seek 完成，10.3版本开始支持。
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	循环播放，一轮播放结束（10.8 版本开始支持）。
VOD_PLAY_EVT_HIT_CACHE	2002	启播时命中缓存事件（11.2 版本开始支持）。
VOD_PLAY_EVT_VIDEO_SEI	2030	收到 SEI 帧事件（ 播放器高级版11.6 版本开始支持）。

SEI 帧

SEI（Supplemental Enhancement Information）帧是一种用于传递附加信息的帧类型，播放器高级版会解析视频流中的 SEI 帧，通过
VOD_PLAY_EVT_VIDEO_SEI 事件回调， 注意：播放器高级版 11.6 版本开始支持。

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:
(NSDictionary*)param {
    if (EvtID == VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = [param objectForKey:EVT_KEY_SEI_TYPE]; //  the type of video SEI
        int seiSize = [param objectForKey:EVT_KEY_SEI_SIZE]; // the data size of video SEI
        NSData *seiData = [param objectForKey:EVT_KEY_SEI_DATA]; // the byte array data
of video SEI
    }
}
```

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	210	硬解启动失败，采用软解。

	6	

连接事件

此外还有几个连接服务器的事件，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_ROATION	2011	MP4 视频旋转角度

视频信息事件

事件 ID	数值	含义说明
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，SDK 会将一些请求信息通知到上层。您可以在收到 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_KEY_WATER_MARK_TEXT	幽灵水印文本内容（11.6 版本开始支持）

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*)param
{
    if (EvtID == 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 (EvtID == PLAY_EVT_PLAY_BEGIN) {
        // 收到开始播放事件
    } else if (EvtID == PLAY_EVT_PLAY_END) {
        // 收到开始结束事件
    }
}
```

幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程 。幽灵水印的内容在收到播放器的 `VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `[param objectForKey:@"EVT_KEY_WATER_MARK_TEXT"]` 获取。详细使用教程参见 超级播放器组件 > 幽灵水印 。

注意： 播放器 11.6 版本开始支持。

播放错误事件

> **说明：**
> [-6004 , -6010] 错误事件 11.0 版本开始支持。

事件 ID	数值	含义说明
PLAY_ERR_NET_DISCONNECT	-2301	视频数据错误导致重试亦不能恢复正常播放。如：网络异常或下载数据错误，导致解封装超时或失败。
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	系统播放器播放错误
VOD_PLAY_ERR_DECODE_VIDEO_FAIL	-6006	视频解码错误，视频格式不支持。
VOD_PLAY_ERR_DECODE_	-6007	音频解码错误，音频格式不支持。

AUDIO_FAIL		
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	字幕解码错误
VOD_PLAY_ERR_RENDER_FAIL	-6009	视频渲染错误
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	视频后处理错误
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	获取点播文件信息失败，建议检查AppId、FileId或Psign填写是否正确。

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
CPU_USAGE	当前瞬时 CPU 使用率
VIDEO_WIDTH	视频分辨率 – 宽
VIDEO_HEIGHT	视频分辨率 – 高
NET_SPEED	当前的网络数据接收速度，单位 KBps。
VIDEO_FPS	当前流媒体的视频帧率
VIDEO_BITRATE	当前流媒体的视频码率，单位 bps。
AUDIO_BITRATE	当前流媒体的音频码率，单位 bps。
V_SUM_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。
SERVER_IP	连接的服务器 IP

通过 onNetStatus 获取视频播放过程信息示例：

```
- (void)onNetStatus:(TXVodPlayer *)player withParam:(NSDictionary *)param {
    //获取当前CPU使用率
    float cpuUsage = [[param objectForKey:@"CPU_USAGE"] floatValue];
    //获取视频宽度
    int videoWidth = [[param objectForKey:@"VIDEO_WIDTH"] intValue];
    //获取视频高度
    int videoHeight = [[param objectForKey:@"VIDEO_HEIGHT"] intValue];
```

```
    //获取实时速率
    int  speed = [[param objectForKey:@"NET_SPEED"] intValue];
    //获取当前流媒体的视频帧率
    int fps = [[param objectForKey:@"VIDEO_FPS"] intValue];
    //获取当前流媒体的视频码率，单位 bps
    int videoBitRate = [[param objectForKey:@"VIDEO_BITRATE"] intValue];
    //获取当前流媒体的音频码率，单位 bps
    int audioBitRate = [[param objectForKey:@"AUDIO_BITRATE"] intValue];
    //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 = [[param objectForKey:@"V_SUM_CACHE_SIZE"] intValue];
    //获取连接的服务器的IP地址
    NSString *ip = [param objectForKey:@"SERVER_IP"];
}
```

其它功能使用

HLS 直播视频源播放

播放器高级版本支持播放 HLS 直播视频源，从 11.8 版本开始支持带 HLS EVENT 直播视频源。用法如下：

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMediaType:MEDIA_TYPE_HLS_LIVE];   // 指定HLS直播媒资类型
[_txVodPlayer setConfig:_config];
[_txVodPlayer startVodPlay:${YOUR_HSL_LIVE_URL}];
```

场景化功能

1、动态设置 AudioSession

有时候需要根据场景来动态设置播放音频输出方式，特别是对于 iPhone 来说，天然支持多音频播放及后台模式。因此，我们根据用户的场景支持了以下三种主要的模式：

- AVAudioSessionCategoryPlayback：后台独占播放。
- AVAudioSessionCategoryPlayAndRecord：后台独占播放。
- AVAudioSessionCategoryAmbient：混合播放。

可以根据所处的场景，利用上面的模式来设置 AudioSession 的 Category 和 Option 来达到自己的目的。下面罗列了两种场景的设置(以下设置可以根据自己的场景进行动态调整和设置)：
场景一：播放列表场景（视频播放需要支持列表里静音播放，并且不打断外部音频播放）。

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback
withOptions:AVAudioSessionCategoryOptionInterruptSpokenAudioAndMixWithOthers
error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
```

场景二：播放详情场景（视频详情有声音，并且暂时打断外部音频，当视频播放完成以后，就恢复外部音频）。

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryAmbient
withOptions:AVAudioSessionCategoryOptionMixWithOthers error:nil];
[[AVAudioSession sharedInstance] setActive:NO
withOptions:AVAudioSessionSetActiveOptionNotifyOthersOnDeactivation error:nil];
```

2、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。

3、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，我们会提供更多的基于用户场景的组件。您可以通过 Player_iOS 下载体验。

直播场景

最近更新时间：2024-05-07 14:22:21

基础知识

本文主要介绍视频云 SDK 的直播播放功能。

直播和点播

- **直播（LIVE）**的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。
- **点播（VOD）**的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放， 播放中您可以通过进度条控制播放位置，腾讯视频和优酷土豆等视频网站上的视频观看就是典型的点播场景。

协议的支持

通常使用的直播协议如下，标准直播推荐使用 FLV 协议的直播地址（以 `http` 开头，以 `.flv` 结尾），快直播使用 WebRTC 协议，更多信息请参见 快直播拉流：

直播协议	优点	缺点	播放延迟
FLV	成熟度高、高并发无压力	需集成 SDK 才能播放	2s – 3s
RTMP	延迟较低	高并发情况下表现不佳	1s – 3s
HLS(m3u8)	手机浏览器支持度高	延迟非常高	10s – 30s
WebRTC	延迟最低	需集成 SDK 才能播放	< 1s

> **注意：**
> - WebRTC 直播协议从播放器 SDK 10.9 版本开始支持。
> - 标准直播与快直播计费价格不同，更多计费详情请参见 标准直播计费 和 快直播计费 。

特别说明

视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP、HLS（m3u8）和 WebRTC 四种格式的直播地址，以及 MP4、 HLS（m3u8）和 FLV 三种格式的点播地址。

对接攻略

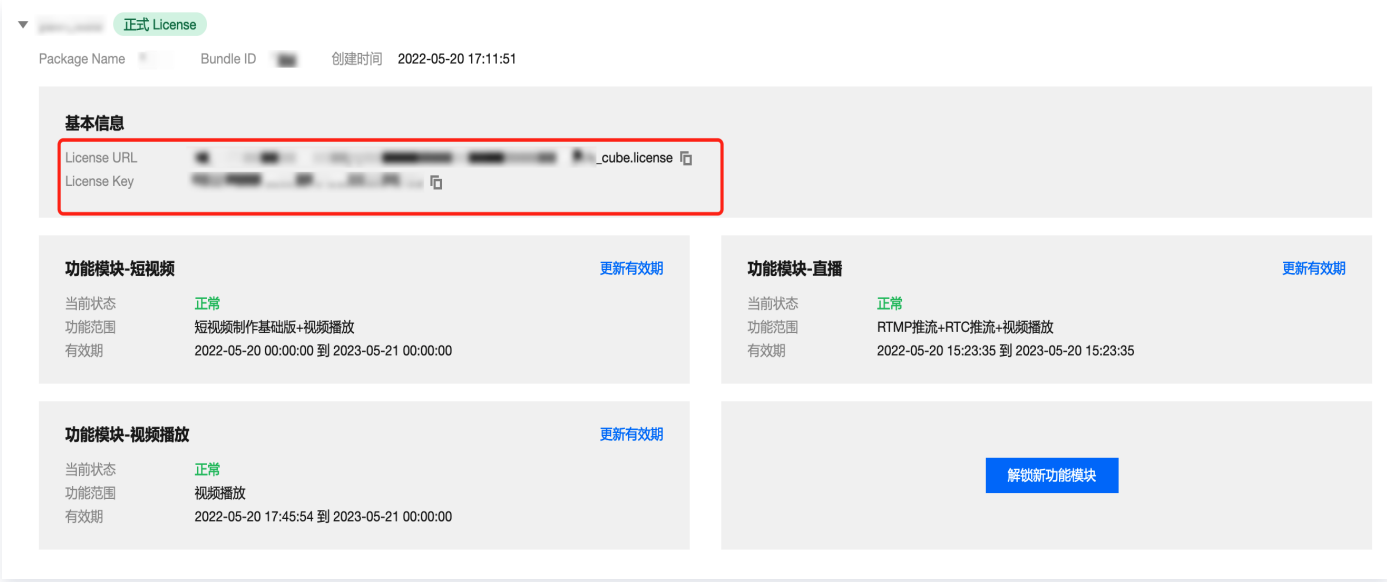
1. 下载 SDK 开发包

下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引 。

2. 给 SDK 配置 License 授权

1. 获取 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。

2. 在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

    //TXLiveBase 位于 "TXLiveBase.h" 头文件中
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
```

> ⚠ **注意：**
> **License 中配置的 BundleId 必须和应用本身一致，否则会播放失败。**

3. 创建 Player

视频云 SDK 中的 V2TXLivePlayer 模块负责实现直播播放功能。

```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
```

4. 渲染 View

接下来我们要给播放器的视频画面找个地方来显示，iOS 系统中使用 view 作为基本的界面渲染单位，所以您只需要准备一个 view 并调整好布局就可以了。

```
//用 setRenderView 给播放器绑定决定渲染区域的view.
[_txLivePlayer setRenderView:_myView];
```

内部原理上，播放器并不是直接把画面渲染到您提供的 view（示例代码中的 `_myView`）上，而是在这个 view 之上创建一个用于 OpenGL 渲染的子视图（subView）。
如果您要调整渲染画面的大小，只需要调整您所常见的 view 的大小和位置即可，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。

如何做动画?
针对 view 做动画是比较自由的，不过请注意此处动画所修改的目标属性应该是 transform 属性而不是 frame 属性。

```
[UIView animateWithDuration:0.5 animations:^{
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
}];
```

5. 启动播放

```
NSString* url = @"http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
[_txLivePlayer startLivePlay:url];
```

6. 画面调整

- **setRenderFillMode：铺满 or 适应**

可选值	含义
V2TXLiveFillModeFill	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
V2TXLiveFillModeFit	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：视频画面顺时针旋转角度**

可选值	含义
V2TXLiveRotation0	不旋转
V2TXLiveRotation90	顺时针旋转90度
V2TXLiveRotation180	顺时针旋转180度
V2TXLiveRotation270	顺时针旋转270度

最长边填充

完全填充

横屏模式

7. 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，这是和点播对比的最大不同点（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
[_txLivePlayer pauseAudio];
[_txLivePlayer pauseVideo];
// 恢复
[_txLivePlayer resumeAudio];
[_txLivePlayer resumeVideo];
```

8. 结束播放

```
// 停止播放
[_txLivePlayer stopPlay];
```

9. 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通过 V2TXLivePlayerObserver 的 onSnapshotComplete 回调截屏图片，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 iOS 的系统 API 来实现。


```
...
[_txLivePlayer setObserver:self];
[_txLivePlayer snapshot];
...

- (void)onSnapshotComplete:(id<V2TXLivePlayer>)player image:(TXImage *)image {
    if (image != nil) {
        dispatch_async(dispatch_get_main_queue(), ^{
            [self handle:image];
        });
    }
}
```

10、HLS 直播视频源播放

HLS 直播视频源播放需要播放器高级版支持， 详细播放教程请 点击这里 。

延时调节

腾讯云 SDK 的直播播放功能，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
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（例如绝地求生）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**

```
//自动模式
[_txLivePlayer setCacheParams:1 maxTime:5];
//极速模式
[_txLivePlayer setCacheParams:1 maxTime:1];
//流畅模式
[_txLivePlayer setCacheParams:5 maxTime:5];

//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，请参见 如何优化视频卡顿 。

SDK 事件监听

您可以为 V2TXLivePlayer 对象绑定一个 V2TXLivePlayerObserver，之后 SDK 的内部状态信息例如播放器状态、播放音量回调、音视频首帧回调、统计数据、警告和错误信息等会通过对应的回调通知给您。

定时触发的状态通知

- onStatisticsUpdate 通知每2秒都会被触发一次，目的是实时反馈当前的播放器状态，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
appCpu	当前 App 的 CPU 使用率（%）
systemCpu	当前系统的 CPU 使用率（%）
width	视频宽度
height	视频高度

fps	帧率（fps）
audioBitrate	音频码率（Kbps）
videoBitrate	视频码率（Kbps）

- onPlayoutVolumeUpdate 播放器音量大小回调。这个回调仅当您调用 enableVolumeEvaluation 开启播放音量大小提示之后才会工作。回调的时间间隔也会与您在设置 enableVolumeEvaluation 的参数 intervalMs 保持一致。

非定时触发的状态通知

其余的回调仅在事件发生时才会抛出来。

Android SDK 集成指引

最近更新时间：2024-04-26 11:00:51

本文主要介绍如何快速地将腾讯云视立方 · 播放器 SDK 集成到您的项目中，不同版本的 SDK 集成方式都通用，按照如下步骤进行配置，就可以完成 SDK 的集成工作。

开发环境要求

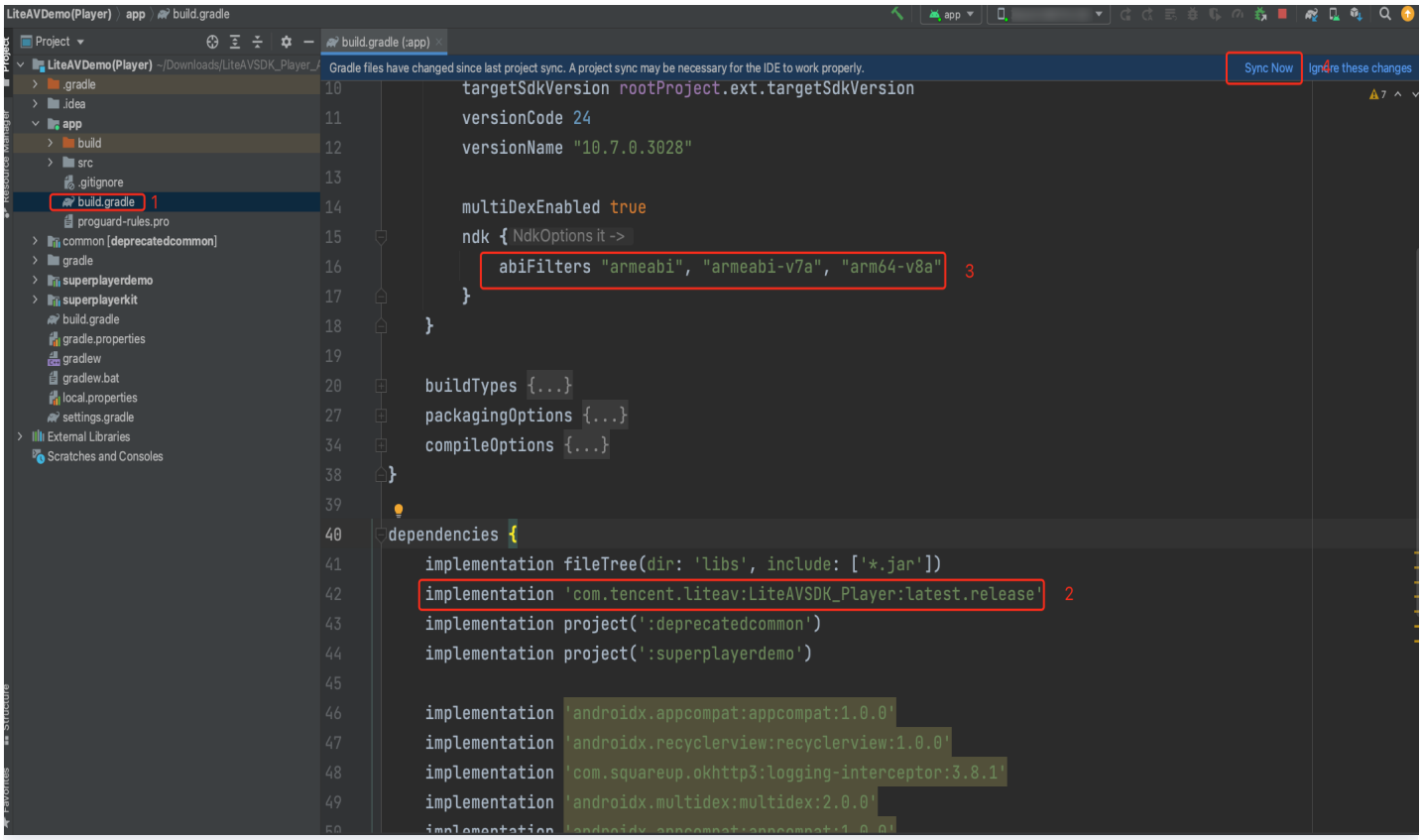
- Android Studio 2.0+。
- Android 4.1（SDK API 16）及以上系统。

集成 SDK（aar）

您可以选择使用 Gradle 自动加载的方式，或者手动下载 aar 再将其导入到您当前的工程项目中。

方法一：自动加载（aar）

播放器 SDK 已经发布到 mavenCentral 库（播放高级版本 mavenCentral 库），您可以通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK_Player。
只需要用 Android Studio 打开需要集成 SDK 的工程，然后通过简单的四个步骤修改 `build.gradle` 文件，就可以完成 SDK 集成：



1. 打开工程根目录下的 build.gradle，添加mavenCentral库。

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

2. 打开 app 下的 build.gradle，在 dependencies 中添加 LiteAVSDK_Player 的依赖。

```
dependencies {
  // 此配置默认集成 LiteAVSDK_Player 最新版本
 implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
  // 集成历史版，如：10.7.0.13038 版本，可通过下面方式集成
  // implementation 'com.tencent.liteav:LiteAVSDK_Player:10.7.0.13038'
}
```

如果您需要集成播放高级版本（Premium）SDK，在 dependencies 中添加如下依赖：

```
 dependencies {
  // 此配置默认集成 LiteAVSDK_Player_Premium 最新版本
 implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
  // 集成历史版，如：10.7.0.13038 版本，可通过下面方式集成
```

```
  // implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:10.7.0.13038'
}
```

3. 在 defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK_Player 支持 armeabi 、 armeabi-v7a 和 arm64-v8a）。

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

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。

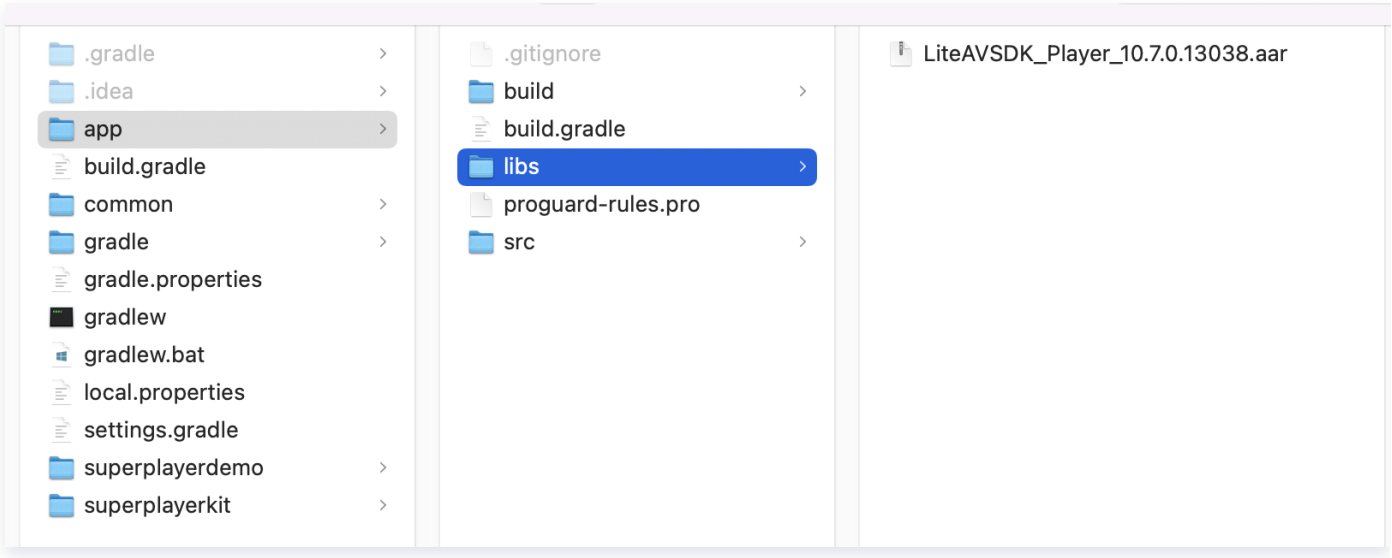
方法二：手动下载（aar）

如果您的网络连接 mavenCentral 有问题，也可以手动下载 SDK 集成到工程里：

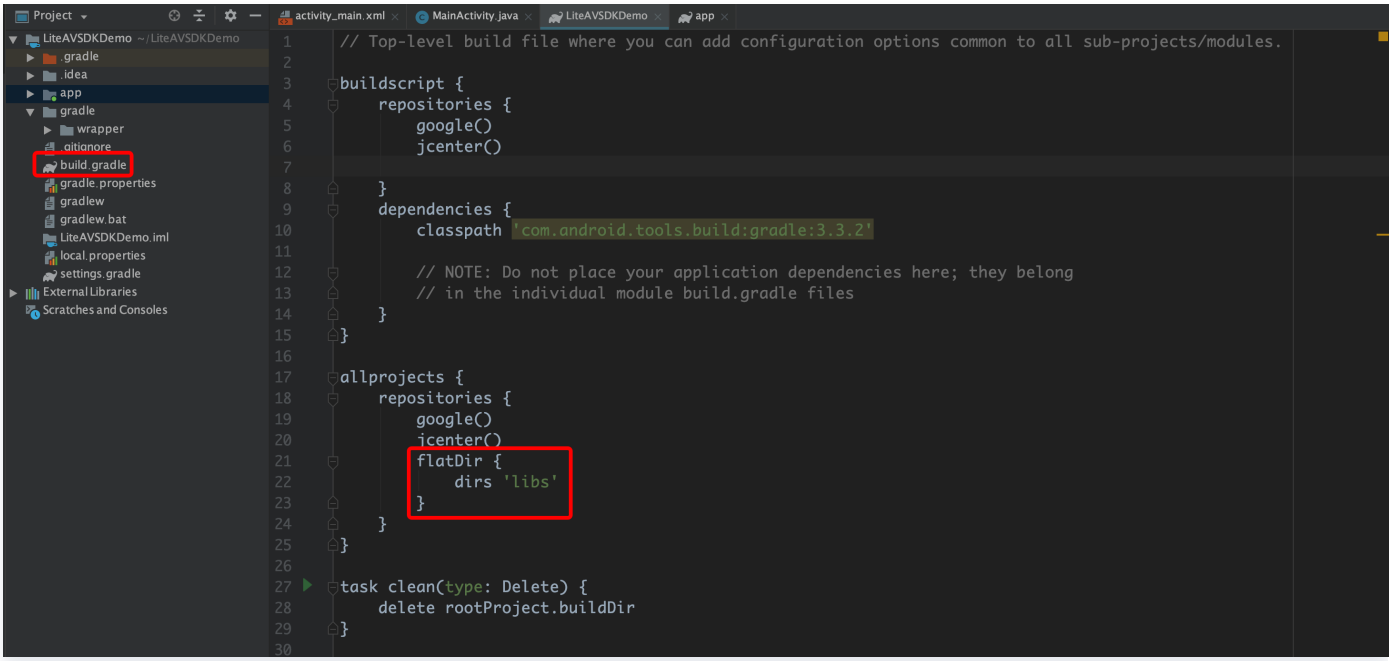
1. 下载 LiteAVSDK_Player ，下载完成后进行解压。

   如果您需要集成播放高级版本（Premium）SDK，请 单击此处 下载。

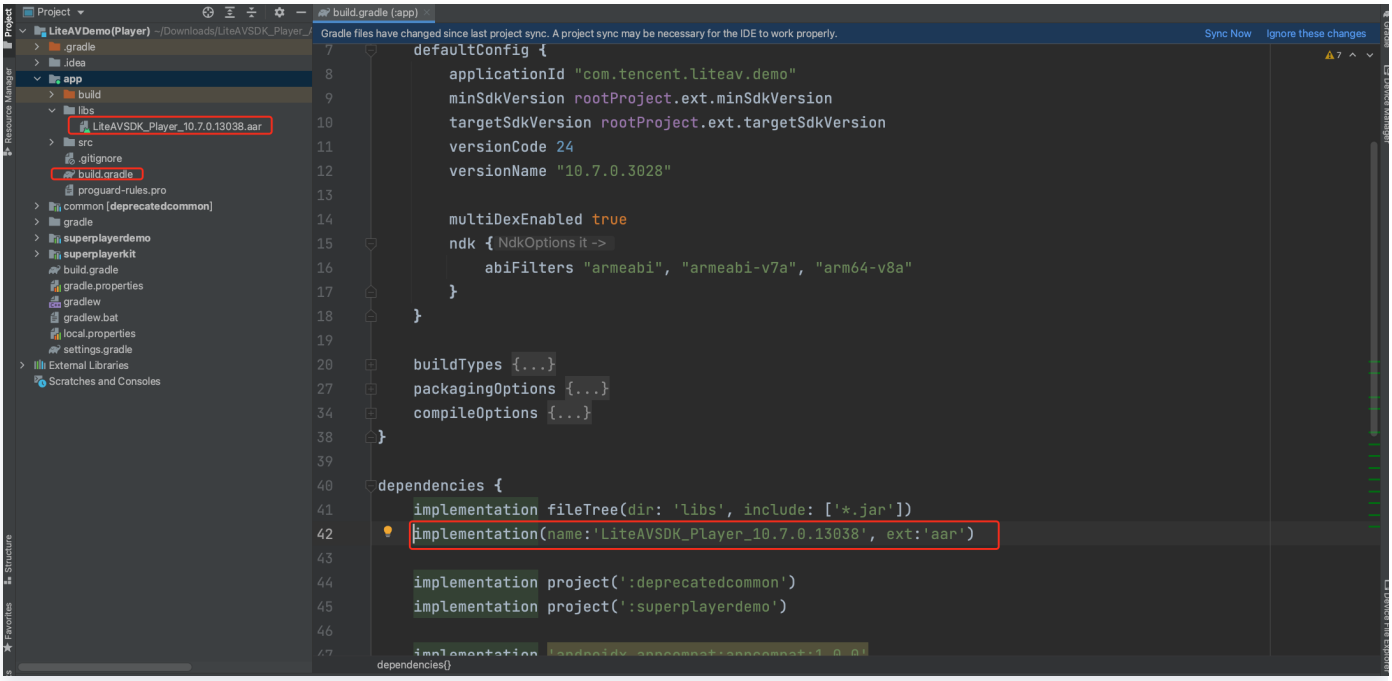
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：



3. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



4. 添加 LiteAVSDK_Player 依赖，在 app/build.gradle 中，添加引用 aar 包的代码。




```
implementation(name:'LiteAVSDK_Player_10.7.0.13038', ext:'aar')
```

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

```
defaultConfig {
 ndk {
```

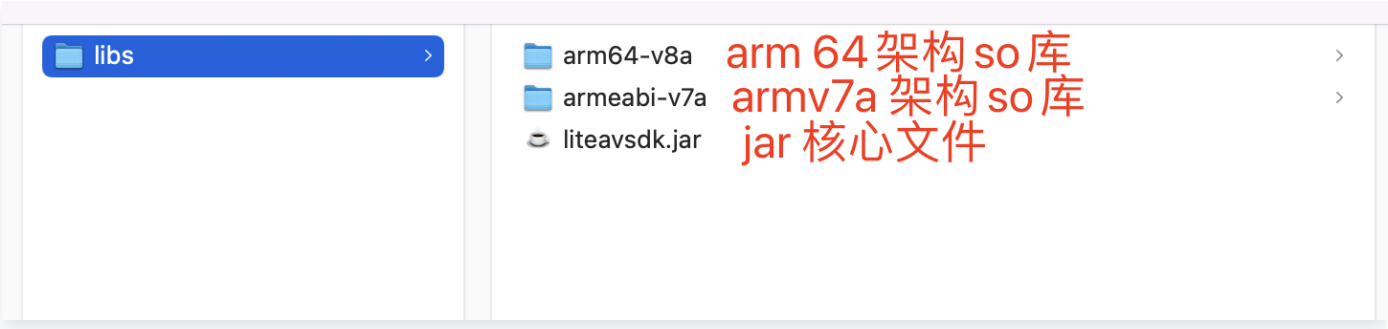
```
    abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
  }
}
```

6. 单击 Sync Now 按钮同步 SDK，完成 LiteAVSDK 的集成工作。

集成 SDK（jar）

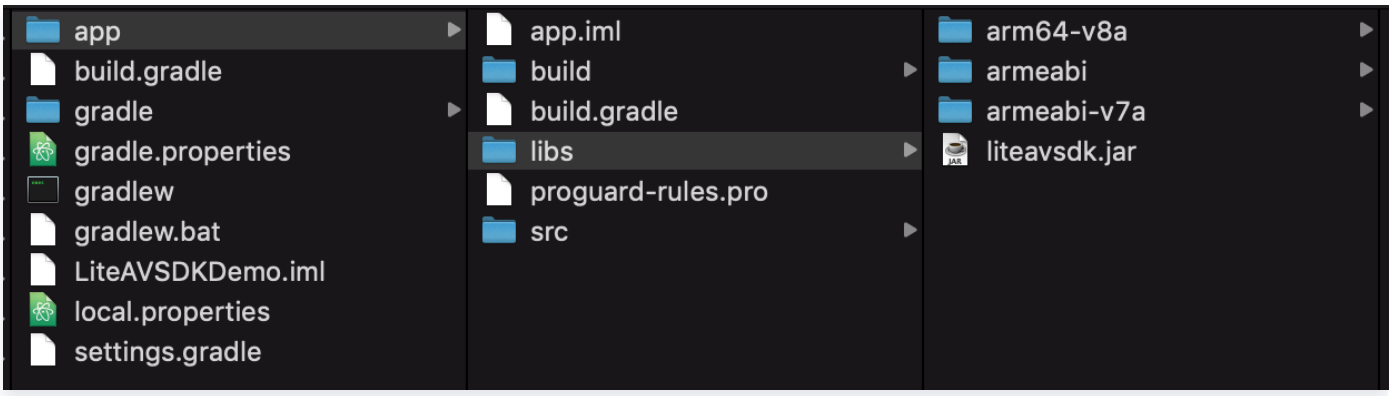
如果您不想集成 aar 库，也可以通过导入 jar 和 so 库的方式集成 LiteAVSDK：

1. 下载 LiteAVSDK_Player ，下载完成后进行解压。在 SDK 目录下找到 LiteAVSDK_Player_xxx.zip （其中 xxx 为 LiteAVSDK 的版本号），解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：

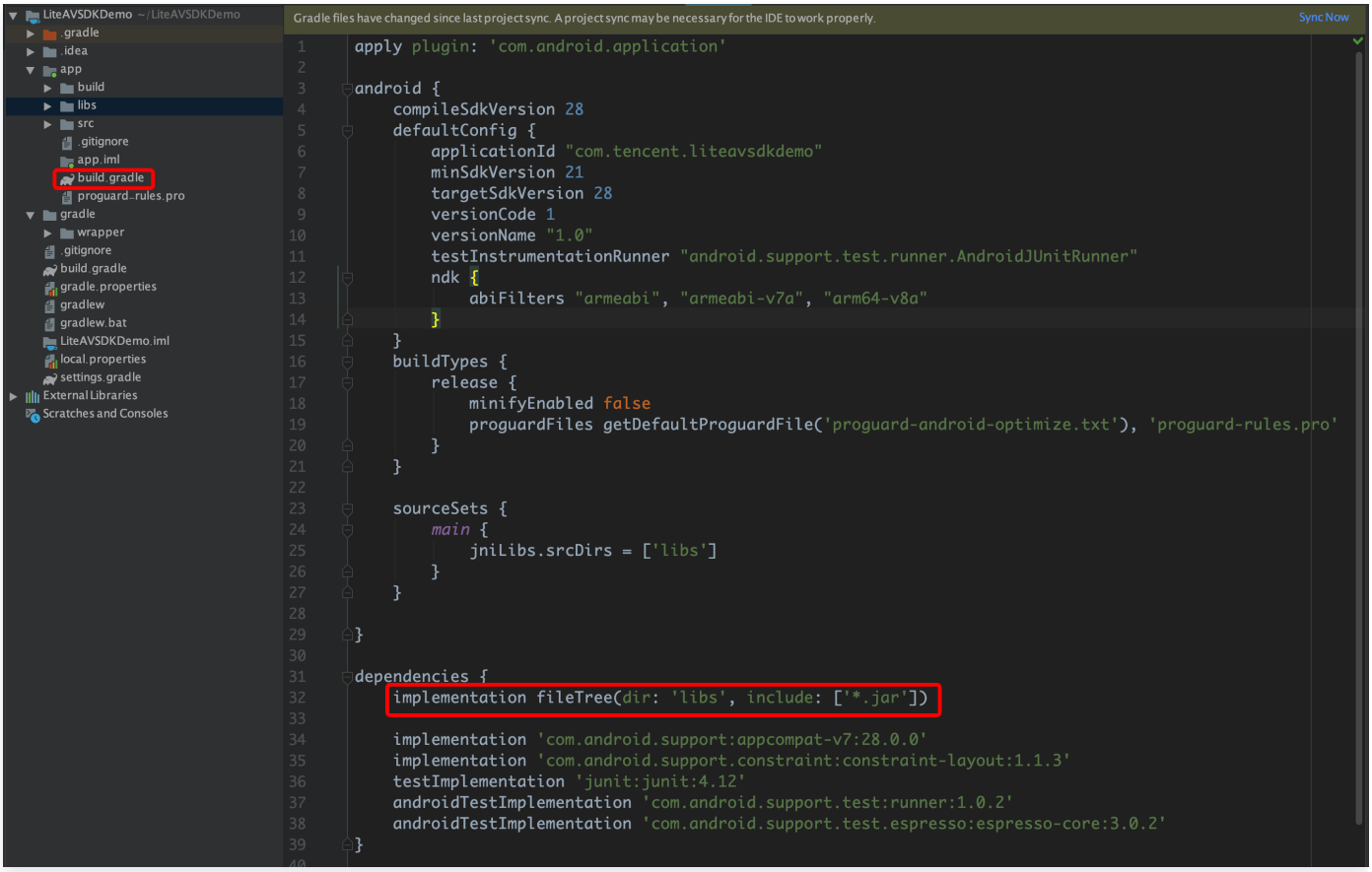


如果您还需要 armeabi 架构 so，复制一份 armeabi-v7a 目录，重命名为 armeabi 即可。
如果您需要集成播放高级版本（Premium）SDK，请 单击此处 下载。

2. 将解压得到的 jar文件和 armeabi、armeabi-v7a、arm64-v8a 文件夹拷贝到 app/libs 目录下。

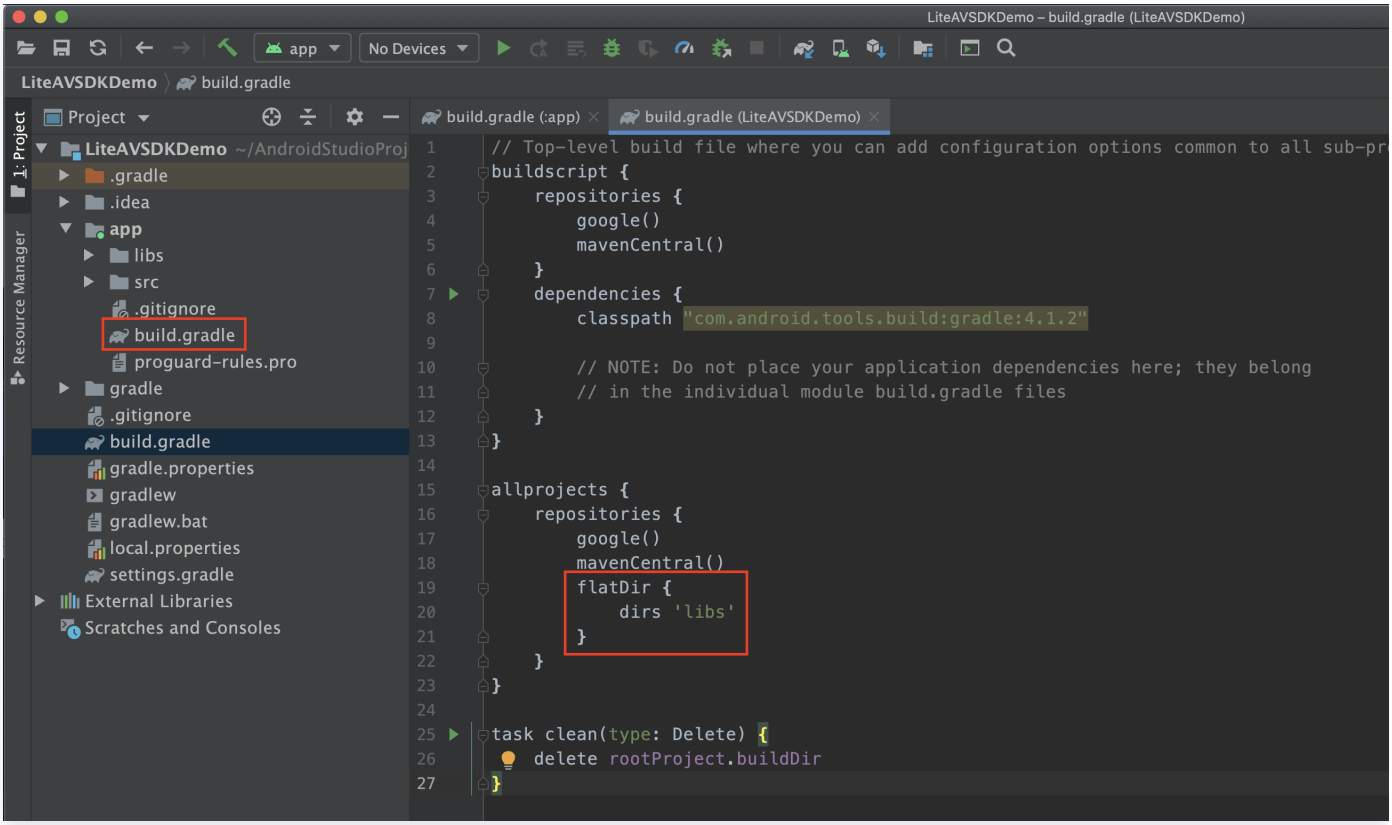


3. 在 app/build.gradle 中，添加引用 jar 库的代码。

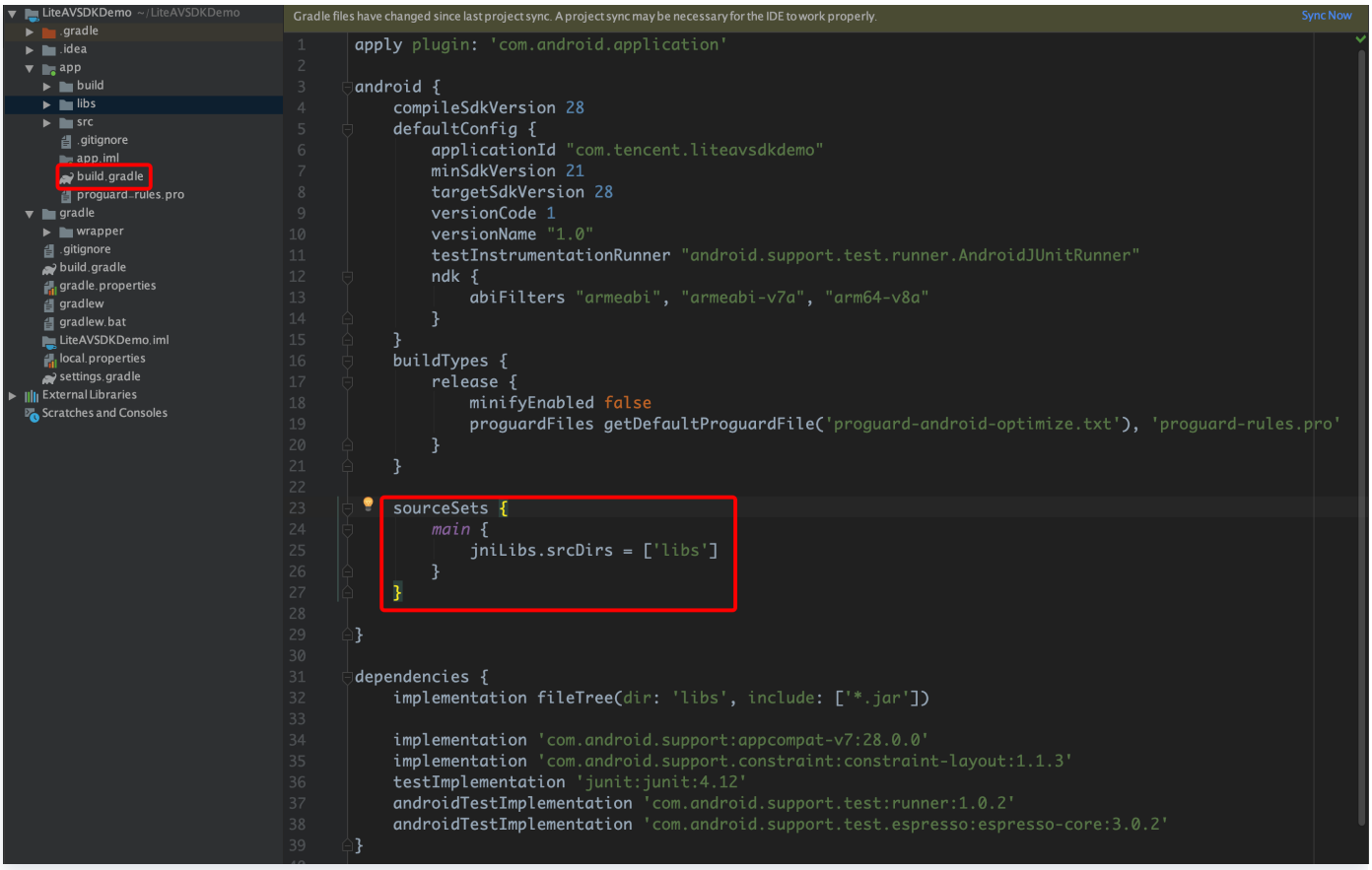



```
dependencies{
 implementation fileTree(dir:'libs',include:['*.jar'])
}
```

4. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



5. 在 app/build.gradle 中，添加引用 so 库的代码。



6. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持

armeabi、armeabi-v7a 和 arm64-v8a）。

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

7. 单击 Sync Now 按钮同步 SDK，完成 LiteAVSDK 的集成工作。

配置 App 打包参数

> **注意：**
> 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：

```
// 新客户集成 SDK 建议把下面的脚本加上，可以优化包体积
packagingOptions {
   exclude "lib/armeabi/libijkhlscache-master.so"
   exclude "lib/armeabi-v7a/libijkhlscache-master.so"
   exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

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" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

网络安全配置允许 App 发送 http 请求

出于安全考虑，从 Android P 开始，Google 要求 App 的请求都使用加密链接。播放器 SDK 会启动一个 localsever 代理 http 请求，如果您的应用 **targetSdkVersion 大于或等于28**，可以通过 网络安全配置 来开启允许向127.0.0.1发送 http 请求。 否则播放时将出现 "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" 错误， 导致无法播放视频。配置步骤如下：

1. 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置。

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

2. 在 AndroidManifest.xml 文件下的 application 标签增加以下属性。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                 ... >
        ...
    </application>
</manifest>
```

设置混淆规则

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

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

配置 License 授权

单击 License 申请 获取测试用 License，不配置 License 将会播放时视频失败，具体操作请参见 测试版 License 。您会获得两个字符串：一个字符串是 licenseURL，另一个字符串是解密 key。
获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License 。

常见问题

项目里面同时集成了直播 SDK/实时音视频/播放器等 LiteAVSDK 系列的多个 SDK 报符号冲突问题怎么解决?

如果集成了2个或以上产品（直播、播放器、TRTC、短视频）的 LiteAVSDK 版本，编译时会出现库冲突问题，因为有些 SDK 底层库有相同符号文件，这里建议只集成一个全功能版 SDK 可以解决，直播、播放器、TRTC、短视频这

些都包含在一个 SDK 里面。具体请参见 SDK 下载 。

点播场景

最近更新时间：2024-05-07 14:22:21

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

- 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。
- 播放器推出短视频组件、画中画2.0、VR 播放等高级组件，功能介绍和使用指引请参见 移动端高级功能 。

SDK 集成

步骤1：集成 SDK 开发包

下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引 。

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 播放器 License 获取相关授权。
- 获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License 。

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 startVodPlay 函数即可。

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


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

// 从 11.8 版本开始支持播放器 content:// URI 视频资源和 asset 目录视频资源
// 播放 content:// URI 视频资源
String localFile = "content://xxx/xxx/video.mp4";
mVodPlayer.startVodPlay(localFile);

// 播放 asset 目录视频资源， 传入的地址必须以 asset:// 开头
String localFile = "asset://video.mp4";
mVodPlayer.startVodPlay(localFile);
```

通过 FileId 方式

```
// 推荐使用下面的新接口
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // 腾讯云账户
的appId
    "4564972819220421305", // 视频的fileId
    "psignxxxxxxx"); // 播放器签名
mVodPlayer.startVodPlay(playInfoParam);

// 旧接口，不推荐使用
TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
authBuilder.setAppId(1252463788);
authBuilder.setFileId("4564972819220421305");
mVodPlayer.startVodPlay(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 startVodPlay 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.startVodPlay(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 默认支持精准 seek，可以在播放器前通过 `TXVodPlayConfig#setEnableAccurateSeek` 进行配置。

```
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);
```

精准和非精准 Seek

播放器 SDK 11.8 版本开始，支持调用 seek 接口时，指定精准或非精准 seek。

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

Seek 到视频流指定 PDT 时间点

跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。
注意：播放器高级版 11.6 版本开始支持。

```
long pdtTimeMs = 600; // 单位为 毫秒
mVodPlayer.seekToPdtTime(time);
```

从指定时间开始播放

首次调用 startVodPlay 之前，支持从指定时间开始播放。

```
float startTimeInSecond = 60; // 单位：秒
mVodPlayer.setStartTime(startTimeInSecond);   // 设置开始播放时间
mVodPlayer.startVodPlay(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，例如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.startVodPlay(url);    // startVodPlay 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。

```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
Map<String, String> headers = new HashMap<>();
headers.put("Referer", "${Refer Content}");
mPlayConfig.setHeaders(headers);
mVodPlayer.setConfig(mPlayConfig);
```

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 **startVodPlay**，否则会产生比较严重的花屏问题。

```
mVodPlayer.stopPlay(true);
mVodPlayer.enableHardwareDecode(true);
mVodPlayer.startVodPlay(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。可以通过下面方法进行清晰度设置。

```
// 获取多码率数组，TXBitrateItem 类字段含义：index-码率下标；width-视频宽；height-视频高；
birate-视频码率
// 在收到播放器 PLAY_EVT_VOD_PLAY_PREPARED 事件调用 getSupportedBitrates 才会有值返
回
ArrayList<TXBitrateItem> bitrates = mVodPlayer.getSupportedBitrates();
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

// 获取当前播放的码率下标，返回值 -1000 为默认值，表示没有设置过码率标；返回值 -1 表示开启
了自适应码流
int index = mVodPlayer.getBitrateIndex();
```

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。

如果您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，从而避免播放后切换码流。详细方法参考 播放器配置#启播前指定分辨率 。

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。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。

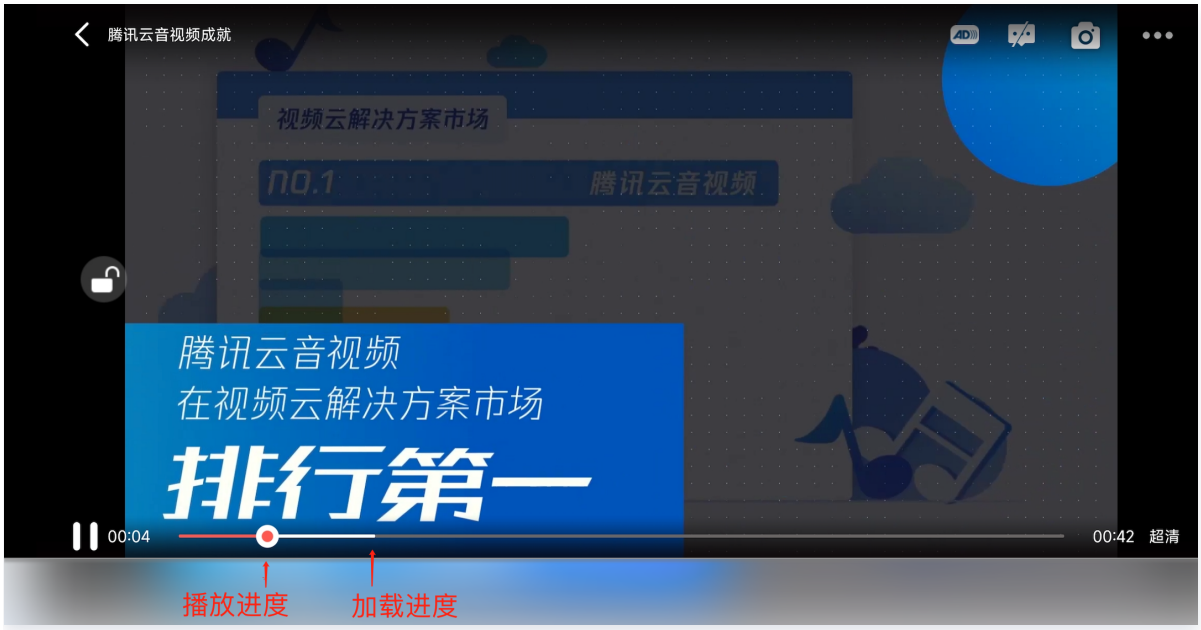
12、开启平滑切换码率

在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。

```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
// 设为true，在IDR对齐时可平滑切换码率, 设为false时，可提高多码率地址打开速度
mPlayConfig.setSmoothSwitchBitrate(true);
mVodPlayer.setConfig(mPlayConfig);
```

13、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

            // 获取 PDT 时间，播放器高级版 11.6 版本开始支持
            long pdt_time_ms = param.getLong(TXVodConstants.EVT_PLAY_PDT_TIME_MS);
        }
    }

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

14、播放网速监听

通过 事件监听 方式，可以在视频播放卡顿时显示当前网速。

- **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);
    }
});
```

15、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();
```

16、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
```

17、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。

18、DRM 加密视频播放

> **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持播放商业级 DRM 加密视频，目前支持 WideVine 和 Fairplay 两种 DRM 方案。更多的商业级 DRM 信息，请参见 商业级 DRM 综述 。
注意：DRM 播放请通过 `TXVodPlayer#setSurface` 配置后播放，否则会出现播放异常。
可通过下面两种方式播放 DRM 加密视频：

通过 FileId 播放

```
// DRM 播放请通过 TXVodPlayer#setSurface 配置后播放，否则会出现播放异常
mVodPlayer.setSurface(mPlayerView.getHolder().getSurface());

// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(${appId}, // 腾讯云账户的
appId
```

```
    ${field}, // DRM 加密视频的 fileId
    ${psgin}); // 加密视频的播放器签名
mVodPlayer.startVodPlay(playInfoParam);
```

通过 FileId 播放适用于接入云点播后台。这种方式和播放普通 FileId 文件没有差别，需要在云点播先配置资源为 DRM 类型，SDK 会在内部识别并处理。

自定义配置播放

```
// DRM 播放请通过 TXVodPlayer#setSurface配置后播放，否则会出现播放异常
mVodPlayer.setSurface(mPlayerView.getHolder().getSurface());

// 步骤一： 设置 Drm 证书提供商环境, 此步骤默认情况下不需要配置
// Google Drm 证书提供商环境默认使用 googleapis.com 域名，可通过下面接口设置为
googleapis.cn 域名
TXPlayerGlobalSetting.setDrmProvisionEnv(TXPlayerGlobalSetting.DrmProvisionEnv.
DRM_PROVISION_ENV_CN);  // googleapis.cn 域名证书地址

// 步骤二: 通过 TXVodPlayer#startPlayDrm 接口播放
TXPlayerDrmBuilder builder = new TXPlayerDrmBuilder();
builder.setPlayUrl(${url});  // 设置播放视频的 url
builder.setKeyLicenseUrl(${keyLicneseUrl});   /// 设置解密 key url
mVodPlayer.startPlayDrm(builder);
```

19、外挂字幕

> **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这两种格式的字幕。
最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `VOD_PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。添加字幕后，并不会主动加载字幕， 调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。
详细用法如下：
步骤1：设置字幕渲染目标对象 View。

```
// 把TXSubtitleView添加到播放器所在布局xml
<com.tencent.rtmp.ui.TXSubtitleView
```

```
        android:id="@+id/subtitle_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

mSubtitleView = findViewById(R.id.subtitle_view);
// 设置字幕渲染目标对象
mVodPlayer.setSubtitleView(mSubtitleView);
```

步骤2：添加外挂字幕。

```
// 传入 字幕url， 字幕名称， 字幕类型， 建议在启动播放器前添加
mVodPlayer.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-
cloud.com/DemoResource/subtitleVTT.vtt", "subtitleName",
TXVodConstants.VOD_PLAY_MIMETYPE_TEXT_VTT);
```

步骤3：播放后切换字幕。

```
// 开始播放视频后，选中添加的外挂字幕， 在收到 VOD_PLAY_EVT_VOD_PLAY_PREPARED 事件
后调用
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VOD_PLAY_PREPARED) {
        List<TXTrackInfo> subtitleTrackInfoList = mVodPlayer.getSubtitleTrackInfo();
        for (TXTrackInfo track : subtitleTrackInfoList) {
            Log.d(TAG, "TrackIndex= " + track.getTrackIndex() + " ,name= " +
track.getName());
            if (TextUtils.equals(track.getName(), "subtitleName")) {
                // 选中字幕
                mVodPlayer.selectTrack(track.trackIndex);
            } else {
                // 其它字幕不需要的话， 进行deselectTrack
                mVodPlayer.deselectTrack(track.trackIndex);
            }
        }
    }
}



// 如果需要，可以监听轨道切换消息
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXVodConstants.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
            int trackIndex = param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_INDEX);
```

```
            int errorCode =
param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_ERROR_CODE);
            Log.d(TAG, "receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=" +
trackIndex + " ,errorCode=" + errorCode);
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle status) {

    }
});
```

步骤4：配置字幕样式。

字幕样式支持在播放前或者播放过程中配置。

```
// 详细参数配置请参考 API 文档
TXSubtitleRenderModel model = new TXSubtitleRenderModel();
model.canvasWidth = 1920;  // 字幕渲染画布的宽
model.canvasHeight = 1080;  // 字幕渲染画布的高
model.fontColor = 0xFFFFFFFF; // 设置字幕字体颜色，默认白色不透明
model.isBondFontStyle = false;  // 设置字幕字体是否为粗体
mVodPlayer.setSubtitleStyle(model);
```

20、多音轨切换

> ⚠ **注意：**
> 此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持切换视频内置的多音轨。用法参见如下代码：

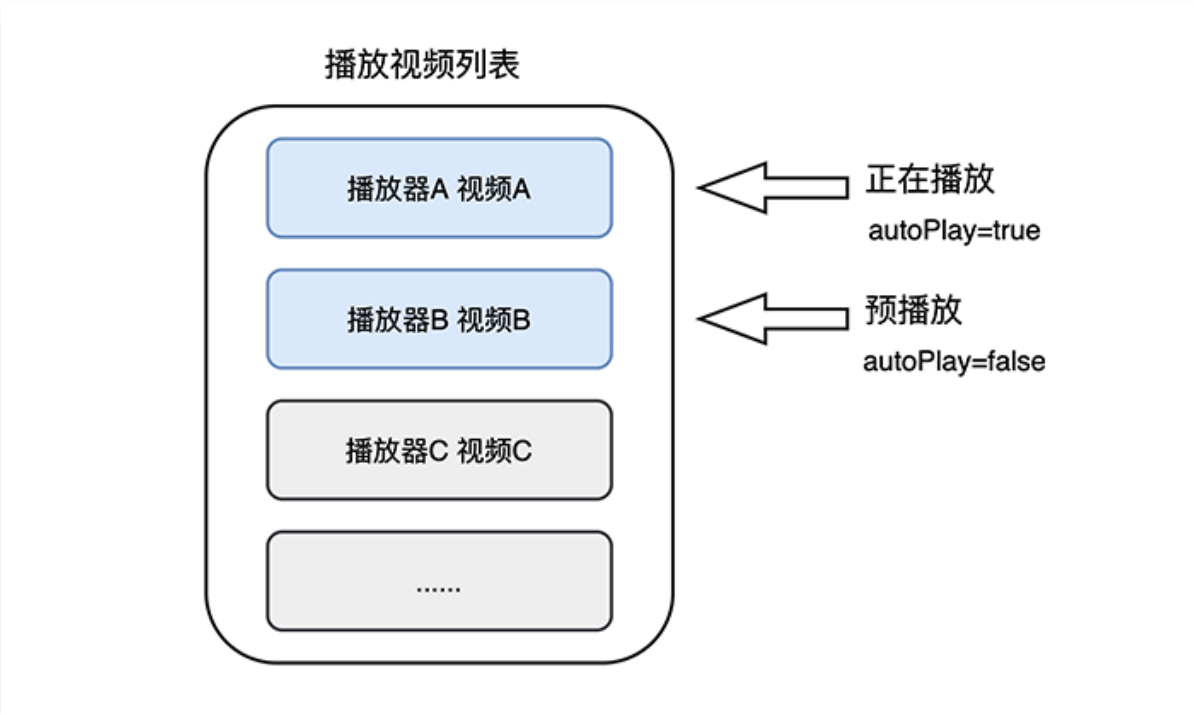
```
// 返回音频轨道信息列表
List<TXTrackInfo> soundTrackInfoList = mVodPlayer.getAudioTrackInfo();
for (TXTrackInfo trackInfo : soundTrackInfoList) {
    if (trackInfo.trackIndex ==  0) {
        // 通过判断 trackIndex 或者 name 切换到需要的音轨
        mVodPlayer.selectTrack(trackInfo.trackIndex);
    } else {
        // 不需要的音轨进行 deselectTrack
        mVodPlayer.deselectTrack(trackInfo.trackIndex);
    }
}
```

进阶功能使用

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 startVodPlay 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.startVodPlay(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.startVodPlay(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：**
> - TXPlayerGlobalSetting 是全局缓存设置接口，原有 TXVodConfig 的缓存配置接口废弃。
> - 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。

通过媒资 URL 预下载

通过媒资 URL 预下载视频代码示例如下：

```
//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(getApplicationContext());
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);
```

通过媒资 FileId 预下载

> **注意：**
> 通过 fileId 预下载从 11.3 版本开始支持。

通过 fileId 预下载是耗时操作， 请不要在主线程调用，否则会抛出非法调用异常。startPreload 时传入的 preferredResolution 要和启播时设置的优先启播分辨率保持一致，否则将达不到预期的效果。 `onStart` 回调的 url 可以保存起来，传给播放器播放。使用示例如下：

```
//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
}

// 启动预下载
Runnable task = new Runnable() {
    @Override
    public void run() {
        TXPlayInfoParams playInfoParams = new TXPlayInfoParams(${appId},
"${fileId}",
                "${psign}");
        // 注意： 耗时操作，请不要在主线程调用！在主线程调用将会抛出非法调用异常。
        mPreLoadManager.startPreload(playInfoParams, 3, 1920 * 1080, new
ITXVodFilePreloadListener() {

            @Override
            public void onStart(int taskID, String fileId, String url, Bundle bundle) {
                // 通过 fileId 换链成功后会回调 onStart， url 可以保存起来，传给播放器播放
                Log.d(TAG, "preload: onStart: taskID: " + taskID + ", fileId: " + fileId + ",
url: " + url + ",bundle: " + bundle);
            }

            @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);
            }
        });
    }
};
new Thread(task).start();
```

```
//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。如果是加密视频，通过播放器 SDK 下载后的视频在本地保持为加密状态，仅可通过腾讯云播放器 SDK 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。
> - 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

SDK 初始化时，设置全局存储路径，用于视频下载，预加载，和缓存等功能。用法如下：

```
File sdcardDir = context.getExternalFilesDir(null);
TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
```

`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();
```

步骤2：开始下载

开始下载有 Fileid 和 URL 两种方式，具体操作如下：

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”。
注意：加密视频只能通过 Fileid 下载，psign 参数必须填写。

```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
```

```
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720, 期望下载此分辨率的流，quality传入 QUALITY_720P)
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
TXVodDownloadDataSource source = new TXVodDownloadDataSource(1252463788, "4564972819220421305", QUALITY_720P, "pSignxxxx", "");  //pSign 加密视频签名，通过fileId下载必填
downloader.startDownload(source);
```

URL 方式

至少需要传入下载地址 URL。preferredResolution 取值为视频分辨率宽和高的乘积：preferredResolution=width * height。如果是嵌套 HLS 格式，preferredResolution 不传入具体值时，默认值为921600。userName 不传入具体值时，默认为"default"。

```
downloader.startDownloadUrl("", 921600 , "");
```

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。

下载错误码

错误码	数值	含义说明
DOWNLOAD_SUCCESS	0	下载成功。
DOWNLOAD_AUTH_FAILED	-5001	向云点播控制台请求视频信息失败，建议检查fileId、psign参数是否正确。
DOWNLOAD_NO_FILE	-5003	无此清晰度文件。
DOWNLOAD_FORMAT_ERROR	-5004	下载文件格式不支持。
DOWNLOAD_DISCONNECT	-5005	网络断开，建议检查网络是否正常。
DOWNLOAD_HLS_KEY_ERROR	-5006	获取 HLS 解密 Key 失败。
DOWNLOAD_PATH_ERROR	-5007	下载目录访问失败，建议检查是否有访问下载目录的权限。
DOWNLOAD_403FORBIDDEN	-5008	请求下载时，鉴权信息不通过，建议检查签名(psign) 是否已过期。

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

1. 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
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
```

```
List<TXVodDownloadMediaInfo> downloadInfoList =
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);
  }
}
```

2. 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
// getDownloadMediaInfo 是耗时接口，请不要在主线程调用
TXVodDownloadMediaInfo downloadInfo =
downloader.getDownloadMediaInfo(1252463788, "4564972819220421305",
QUALITYHD);
// 获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。
// 备注：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大小，暂时无法获取。
long size = downloadInfo.getSize();  // 获取下载文件总大小
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
```

3. 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， 耗时接口，请不要在主线程调用
TXVodDownloadMediaInfo downloadInfo =
downloader.getDownloadMediaInfo("http://1253131631.vod2.myqcloud.com/26f327f9vodgzp1253131631/f4bdff799031868222924043041/playlist.m3u8”);
```

4. 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);
```

步骤6：下载后离线播放

下载后的视频支持无网络的情况下进行播放，无需进行联网。下载完成后，通过 `TXVodDownloadMediaInfo#getPlayPath` 获取到下载地址即可进行播放。

```
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
List<TXVodDownloadMediaInfo> mediaInfoList =
TXVodDownloadManager.getInstance().getDownloadMediaInfoList();
// 业务侧根据实际需求查找到需要播放的 media 对象
for (TXVodDownloadMediaInfo mediaInfo : mediaInfoList) {
    if (mediaInfo.getDownloadState() == TXVodDownloadMediaInfo.STATE_FINISH) {  // 判
断是否下载完成
        mVodPlayer.startVodPlay(mediaInfo.getPlayPath());  // 播放已经下载完成的视频
    }
}
```

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。
在腾讯云控制台提取到 appId， 加密视频的 fileId 和 psign 后，可以通过下面的方式进行播放：

```
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // 腾讯云账户的
appId
    "4564972819220421305", // 视频的fileId
    "psignxxxxxxx"); // 播放器签名
mVodPlayer.startVodPlay(playInfoParam);
```

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
```

- **启播前指定分辨率**

播放 HLS 的多码率视频源，如果您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(TXVodConstants.VIDEO_RESOLUTION_720X1280);
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

- **启播前指定媒资类型**

当提前知道播放的媒资类型时，可以通过配置 `TXVodPlayConfig#setMediaType` 减少播放器SDK内部播放类型探测，提升启播速度。

> **注意：**
> `TXVodPlayConfig#setMediaType` 11.2 版本开始支持。

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMediaType(TXVodConstants.MEDIA_TYPE_FILE_VOD);  // 用于提升MP4启播速度
// config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_VOD);  // 用于提升HLS启播速度
mVodPlayer.setConfig(config);
```

- **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
```

6、HttpDNS 解析服务

移动解析（HTTPDNS）基于 HTTP 协议向 DNS 服务器发送域名解析请求，替代了基于 DNS 协议向运营商 Local DNS 发起解析请求的传统方式，可避免 Local DNS 造成域名劫持和跨网访问问题，解决移动互联网服务中域名解析异常带来的视频播放失败困扰。

> **注意：**
> HttpDNS 解析服务从 10.9 版本开始支持。

1. 开通 HTTPDNS 解析服务

   您可以选择腾讯云或其它云提供商，开通 HTTPDNS 解析服务，确保开通成功后， 再集成到播放 SDK。
2. 在播放 SDK 接入 HTTPDNS 解析服务

下面以接入 腾讯云 HTTPDNS 为例子，展示如何在播放器 SDK 接入：

```
// 步骤1： 打开 HttpDNS 解析开关
TXLiveBase.enableCustomHttpDNS(true);
// 步骤2： 设置 HttpDNS 解析回调， TXLiveBaseListener#onCustomHttpDNS
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onCustomHttpDNS(String hostName, List<String> ipList) {
        // 把 hostName 解析到 ip 地址后，保存到 iPList，返回给 SDK 内部。注意：这里不要进行
异步操作。
        // MSDKDnsResolver 是腾讯云提供的 HTTPDNS SDK 解析接口
        String ips = MSDKDnsResolver.getInstance().getAddrByName(hostname);
        String[] ipArr = ips.split(";");
        if (0 != ipArr.length) {
            for (String ip : ipArr) {
                if ("0".equals(ip)) {
                    continue;
                }
                ipList.add(ip);
            }
        }
    }
});
```

7、HEVC 自适应降级播放

播放器支持同时传入 HEVC 和其它视频编码格式例如： H.264 的播放链接， 当播放机型不支持 HEVC 格式时，将自动降级为配置的其它编码格式（如： H.264 ）的视频播放。
注意：播放器高级版 11.7 版本开始支持。

```
//设置备选播放链接
String backupPlayUrl = "${backupPlayUrl}";  // 备选播放链接
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_MIMETYPE,
TXVodConstants.VOD_PLAY_MIMETYPE_H265); // 指定原始 HEVC 视频编码类型
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_BACKUP_URL, backupPlayUrl); //
设置 H.264 格式等备选播放链接地址

// 设置原始 HEVC 播放链接
String hevcPlayUrl = "${hevcPlayUrl}";
mVodPlayer.startVodPlay(hevcPlayUrl);
```

8、音量均衡

播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。通过 `TXVodPlayer#setAudioNormalization` 设置音量均衡，响度范围：-70～0

(LUFS)，同时支持自定义数值。
注意：播放器高级版 11.7 版本开始支持。

```
/**
 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h）
 关：AUDIO_NORMALIZATION_OFF
 开：AUDIO_NORMALIZATION_STANDARD（标准）
    AUDIO_NORMALIZATION_LOW （低）
    AUDIO_NORMALIZATION_HIGH（高）
 可填自定义数值：从低到高，范围-70 - 0 LUFS
*/
mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD)
; //启动音量均衡

mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_OFF);       //
关闭音量均衡
```

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_END	2006	视频播放结束。
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。
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	循环播放，一轮播放结束（10.8 版本开始支持）。
TXVodConstants.VOD_PLAY_EVT_HIT_CACHE	2002	启播时命中缓存事件（11.2 版本开始支持）。

TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI	2030	收到 SEI 帧事件（播放器高级版11.6 版本开始支持）。

SEI 帧

SEI（Supplemental Enhancement Information）帧是一种用于传递附加信息的帧类型，播放器高级版会解析视频流中的 SEI 帧，通过
VOD_PLAY_EVT_VIDEO_SEI 事件回调， 注意：播放器高级版 11.6 版本开始支持。

```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = param.getInt(TXVodConstants.EVT_KEY_SEI_TYPE);  //  the type of
video SEI
        int seiSize = param.getInt(TXVodConstants.EVT_KEY_SEI_SIZE);  // the data size of
video SEI
        byte[] seiData = param.getByteArray(TXVodConstants.EVT_KEY_SEI_DATA);  // the
byte array data of video SEI
    }
}
```

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：startVodPlay(TXPlayInfoParams playInfoParams)），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_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。
TXVodConstants.EVT_KEY_WATER_MARK_TEXT	幽灵水印文本内容（11.6 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) {
    }
});
```

幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程 。幽灵水印的内容在收到播放器的 `TXVodConstants#VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `param.getString(TXVodConstants.EVT_KEY_WATER_MARK_TEXT)` 获取。详细使用教程参见 超级播放器组件 > 幽灵水印 。

注意： 播放器 11.6 版本开始支持。

播放错误事件

> **说明：**
> [-6004 , -6010] 错误事件 11.0 版本开始支持。

事件 ID	数值	含义说明
PLAY_ERR_NET_DISCONNECT	-2301	视频数据错误导致重试亦不能恢复正常播放。如：网络异常或下载数据错误，导致解封装超时或失败。
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败。
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	系统播放器播放错误。
VOD_PLAY_ERR_DECODE_VI	-6006	视频解码错误，视频格式不支持。

DEO_FAIL		
VOD_PLAY_ERR_DECODE_AUDIO_FAIL	-6007	音频解码错误，音频格式不支持。
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	字幕解码错误。
VOD_PLAY_ERR_RENDER_FAIL	-6009	视频渲染错误。
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	视频后处理错误。
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	获取点播文件信息失败，建议检查AppId、FileId或Psign填写是否正确。

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，单位：KBps。
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 bps。
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 bps。
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 bps
        int videoBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_BITRATE);
        //获取当前流媒体的音频码率，单位 bps
        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);
    }
});
```

其它功能使用

HLS 直播视频源播放

播放器高级版本支持播放 HLS 直播视频源，从 11.8 版本开始支持带 HLS EVENT 直播视频源。用法如下：

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_LIVE); // 指定HLS直播媒资类型
mVodPlayer.setConfig(config);
mVodPlayer.startVodPlay(${YOUR_HSL_LIVE_URL});
```

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，我们会提供更多的基于用户场景的组件。您可以通过 Player_Android 下载体验。

直播场景

最近更新时间：2024-05-07 14:22:21

基础知识

本文主要介绍视频云 SDK 的直播播放功能。

直播和点播

- **直播（LIVE）**的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。
- **点播（VOD）**的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放， 播放中您可以通过进度条控制播放位置，腾讯视频和优酷、土豆等视频网站上的视频观看就是典型的点播场景。

协议的支持

通常使用的直播协议如下，标准直播推荐使用 FLV 协议的直播地址（以 `http` 开头，以 `.flv` 结尾），快直播使用 WebRTC 协议，更多信息请参见 快直播拉流：

直播协议	优点	缺点	播放延迟
FLV	成熟度高、高并发无压力	需集成 SDK 才能播放	2s – 3s
RTMP	延迟较低	高并发情况下表现不佳	1s – 3s
HLS(m3u8)	手机浏览器支持度高	延迟非常高	10s – 30s
WebRTC	延迟最低	需集成 SDK 才能播放	< 1s

说明：

- WebRTC 直播协议从播放器SDK10.9 版本开始支持。
- 标准直播与快直播计费价格不同，更多计费详情请参见 标准直播计费 和 快直播计费 。

特别说明

视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP、HLS（m3u8）和 WebRTC 四种格式的直播地址，以及 MP4、 HLS（m3u8）和 FLV 三种格式的点播地址。

对接攻略

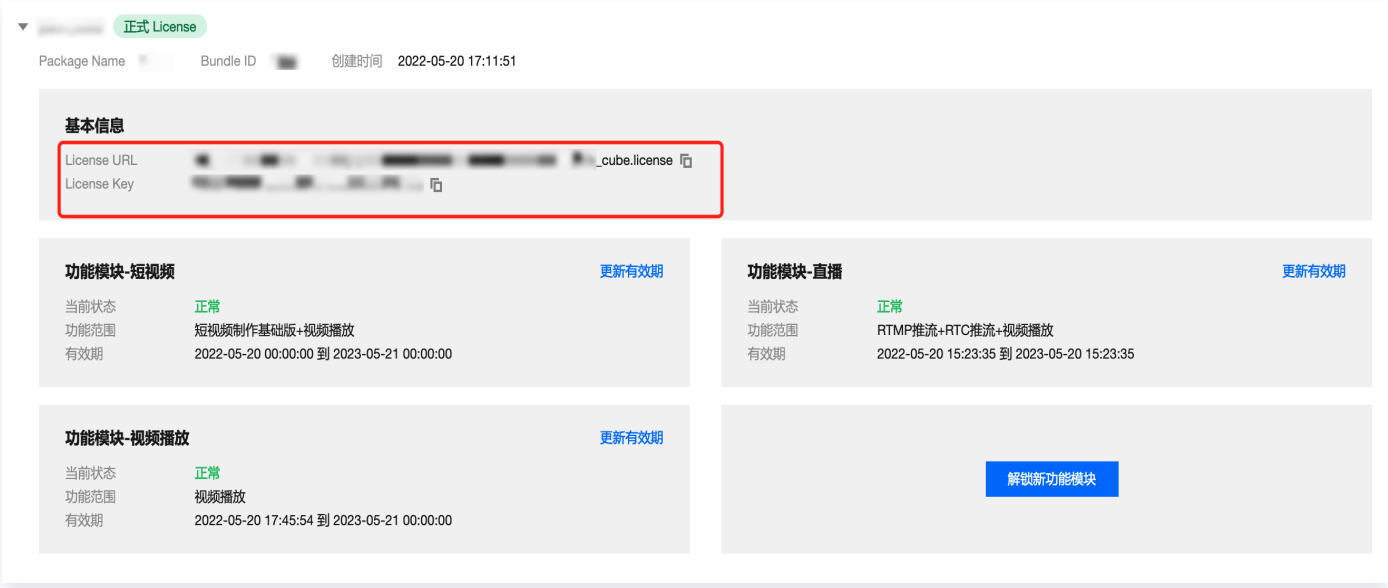
1. 下载 SDK 开发包

下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引 。

2. 给 SDK 配置 License 授权

1. 获取 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。

2. 在您的 App 调用 SDK 相关功能之前（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);
        }
    });
 }
}
```

> **注意**
> License 中配置的 packageName 必须和应用本身一致，否则会播放失败。

3. 添加渲染 View

为了能够展示播放器的视频画面，我们第一步要做的就是在布局 xml 文件里加入渲染 View：

```
<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="visible"/>
```

4. 创建 Player

视频云 SDK 中的 **V2TXLivePlayer** 模块负责实现直播播放功能，并使用 **setRenderView** 接口将它与我们刚刚添加到界面上的 **video_view** 渲染控件进行关联。

```
//mPlayerView 即 step3 中添加的界面 view
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);
//创建 player 对象
V2TXLivePlayer mLivePlayer = new V2TXLivePlayerImpl(mContext);
//关键 player 对象与界面 view
mLivePlayer.setRenderView(mView);
```

5. 启动播放

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startLivePlay(flvUrl);
```

6. 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 step3 中添加的 `video_view` 控件的大小和位置即可。
- **setRenderFillMode：铺满 or 适应**

可选值	含义
V2TXLiveFillModeFill	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
V2TXLiveFillModeFit	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：视频画面顺时针旋转角度**

可选值	含义
V2TXLiveRotation0	不旋转
V2TXLiveRotation90	顺时针旋转90度
V2TXLiveRotation180	顺时针旋转180度
V2TXLiveRotation270	顺时针旋转270度

```
// 设置填充模式
mLivePlayer.setRenderFillMode(V2TXLiveFillModeFit);
// 设置画面渲染方向
mLivePlayer.setRenderRotation(V2TXLiveRotation0);
```

最长边填充

完全填充

横屏模式

7. 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，这是和点播对比的最大不同点（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.pauseAudio();
mLivePlayer.pauseVideo();
// 继续
mLivePlayer.resumeAudio();
mLivePlayer.resumeVideo();
```

8. 结束播放

结束播放非常简单，直接调用 `stopPlay` 即可。

```
mLivePlayer.stopPlay();
```

9. 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.setObserver(new MyPlayerObserver());
mLivePlayer.snapshot();
// 在MyPlayerObserver的回调接口onSnapshotComplete中获取屏幕截图
```

```
private class MyPlayerObserver extends V2TXLivePlayerObserver  {
    ...
    @Override
    public void onSnapshotComplete(V2TXLivePlayer v2TXLivePlayer, Bitmap bitmap) {
    }
    ...
}
```

10、HLS 直播视频源播放

HLS 直播视频源播放需要播放器高级版支持， 详细播放教程请 点击这里 。

延时调节

腾讯云 SDK 的云直播播放功能，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
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（例如绝地求生）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**

```
//自动模式
mLivePlayer.setCacheParams(1.0f, 5.0f);
//极速模式
mLivePlayer.setCacheParams(1.0f, 1.0f);
//流畅模式
mLivePlayer.setCacheParams(5.0f, 5.0f);
//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 如何优化视频卡顿 。

SDK 事件监听

您可以为 V2TXLivePlayer 对象绑定一个 V2TXLivePlayerObserver，之后 SDK 的内部状态信息例如播放器状态、播放音量回调、音视频首帧回调、统计数据、警告和错误信息等会通过对应的回调通知给您。

定时触发的状态通知

- onStatisticsUpdate 通知每2秒都会被触发一次，目的是实时反馈当前的播放器状态，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
appCpu	当前 App 的 CPU 使用率（%）
systemCpu	当前系统的 CPU 使用率（%）
width	视频宽度
height	视频高度
fps	帧率（fps）
audioBitrate	音频码率（Kbps）
videoBitrate	视频码率（Kbps）

- onPlayoutVolumeUpdate 播放器音量大小回调。这个回调仅当您调用 enableVolumeEvaluation 开启播放音量大小提示之后才会工作。回调的时间间隔也会与您在设置 `enableVolumeEvaluation` 的参数 `intervalMs` 保持一致。

非定时触发的状态通知

其余的回调仅在事件发生时才会抛出来。

Flutter SDK 集成指引

最近更新时间：2024-04-26 11:00:52

环境准备

- Flutter 3.0 及以上版本。
- Android 端开发：
  - Android Studio 3.5及以上版本。
  - App 要求 Android 4.1及以上版本设备。
- iOS 端开发：
  - Xcode 11.0及以上版本。
  - OSX 系统版本要求 10.11 及以上版本。
  - 请确保您的项目已设置有效的开发者签名。

SDK 下载

腾讯云视立方 Flutter 播放器项目的地址是 Player Flutter 。

> ⚠ **注意：**
> **运行该 demo 的时候，需要在 demo_config 中设置自己的播放器 license，并在 Android 和 iOS 配置中，将包名和 bundleId 修改为自己签名的包名和 bundleId。**

快速集成

在项目的 pubspec.yaml 中添加依赖

支持基于 LiteAVSDK Player 或 Professional 版本集成，您可以根据项目需要进行集成。

1. 集成 LiteAVSDK_Player 版本最新版本，默认情况下也是集成此版本。在 pubspec.yaml 中增加配置：

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
```

集成 LiteAVSDK_Player_Premium (播放器高级版) 最新版本，则 pubspec.yaml 中配置改为：

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
```

```
    path: Flutter
    ref: Player_Premium
```

集成 LiteAVSDK_Professional 最新版本，则 pubspec.yaml 中配置改为：

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: Professional
```

如果需要集成指定播放器版本的 SDK，可以指定通过 ref 依赖的 tag 来指定到对应版本，如下所示：

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: release_player_v1.0.6

# release_player_v1.0.6 表示将集成Android端TXLiteAVSDK_Player_10.6.0.11182 版本，
iOS 端集成 TXLiteAVSDK_Player_10.6.11821 版本
```

更多归档的 tag 请参考 release 列表 。

2. 集成之后，可以通过代码编辑器自带的 UI 界面来获取 Flutter 依赖，也可以直接使用如下命令获取：

```
flutter pub get
```

3. 使用过程中，可以通过以下命令来更新现有 Flutter 依赖：

```
flutter pub upgrade
```

添加原生配置

Android 端配置

1. 在 Android 的 AndroidManifest.xml 中增加如下配置：

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

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

网络安全配置允许 App 发送 HTTP 请求

出于安全考虑，从 Android P 开始，Google 要求 App 的请求都使用加密链接。播放器 SDK 会启动一个 localsever 代理 HTTP 请求，如果您的应用 **targetSdkVersion 大于或等于28**，可以通过 网络安全配置 来开启允许向127.0.0.1发送 HTTP 请求。 否则播放时将出现 "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" 错误， 导致无法播放视频。配置步骤如下：

1.1 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置。

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

1.2 在 AndroidManifest.xml 文件下的 application 标签增加以下属性。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                 ... >
        ...
    </application>
</manifest>
```

2. 确保 Android 目录下的 `build.gradle` 使用了 mavenCenter，能够成功下载到依赖。

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

3. 配置 Android 最小 SDK 版本，由于 flutter 默认配置的 Android 最小版本过低，需要手动更改为至少19，如果需要使用画中画能力，compileSdkVersion 和targetSdkVersion 则需要修改为至少31。

```
compileSdkVersion 31
defaultConfig {
    applicationId "com.tencent.liteav.demo"
    minSdkVersion 19
    targetSdkVersion 31
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

4. AndroidManifest.xml 根节点 manifest 标签内增加如下配置 xmlns:tools="http://schemas.android.com/tools"，示例如下：

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.player">
 <!-- your config..... -->
</manifest>
```

application 节点下增加 tools:replace="android:label"，示例如下：

```
<application
    android:label="super_player_example"
    android:icon="@mipmap/ic_launcher"
    android:requestLegacyExternalStorage="true"
    tools:replace="android:label">
<!-- your config..... -->
</application>
```

5. 如果需要更新原生 SDK 依赖版本，可手动删除 Android 目录下的 build 文件夹，也可以使用如下命令强制刷新。

```
./gradlew build
```

iOS 端配置

> ⚠ **注意：**
> iOS 端目前暂不支持模拟器运行调试，建议在真机下进行开发调试。

1. 在 iOS 的 Info.plist 中增加如下配置：

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

2. iOS 原生采用 pod 方式进行依赖，编辑 podfile 文件，指定您的播放器 SDK 版本，默认集成的是 Player 版

SDK。

```
pod 'TXLiteAVSDK_Player'          //Player版
```

Professional 版 SDK 集成：

```
pod 'TXLiteAVSDK_Professional'   //Professional版
```

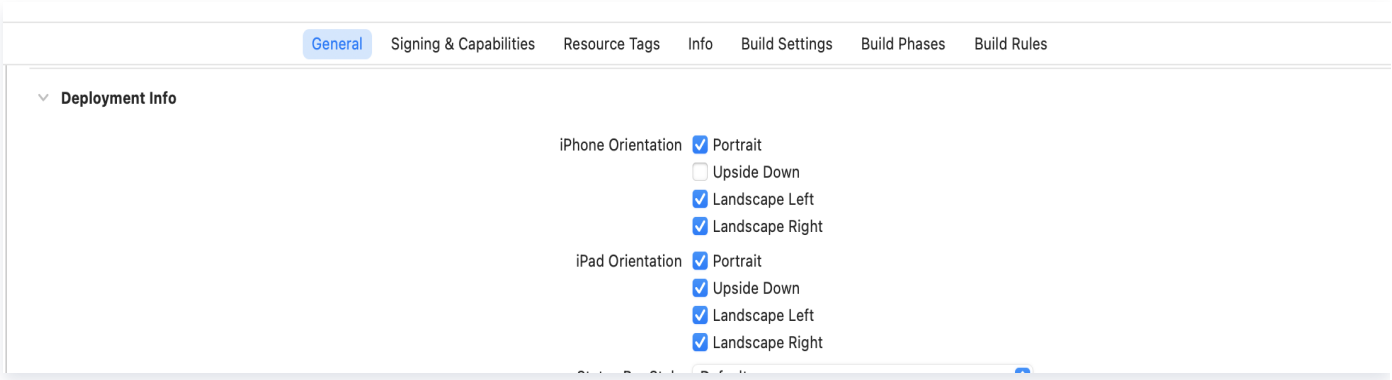
如果不指定版本，默认会安装最新的 `TXLiteAVSDK_Player` 最新版本。

3. 部分情况下（如：发布了新版本），需要强制更新 iOS 播放器依赖，可以在 iOS 目录下使用如下命令进行更新：

```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

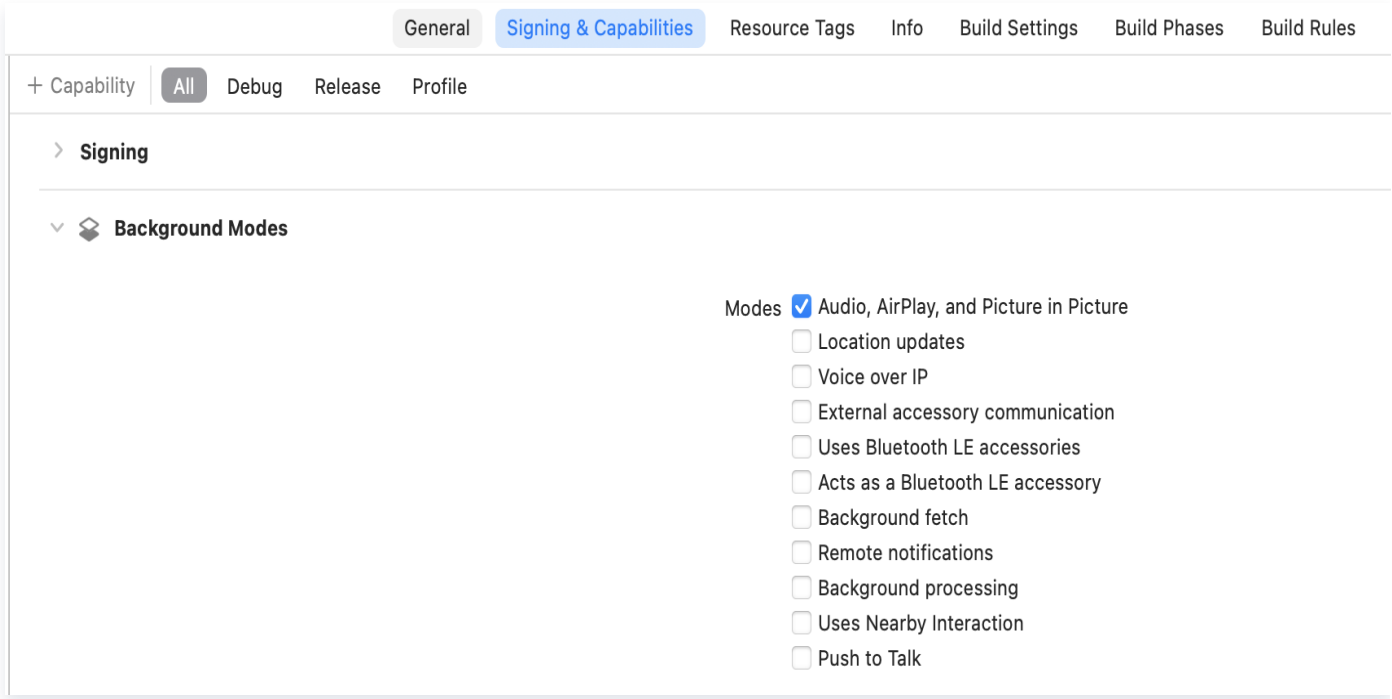
4. 横屏配置

如果需要应用支持横屏，需要在 iOS 项目配置 `General` 页面的 `Deployment Info` 标签下设置横竖屏的支持方向，可以全部进行勾选，如下图所示：



5. 画中画配置

如果需要项目支持画中画，需要在 iOS 项目配置 `Signing & Capabilities` 页面的 `Background Modes` 标签下，勾选 "Audio, AirPlay, and Picture in Picture" 来让项目支持画中画能力，如下图所示：



集成播放器 License

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 播放器 License 获取相关授权。
集成播放器前，需要 注册腾讯云账户 ，注册成功后申请播放器 License， 然后通过下面方式集成，建议在应用启动时进行。
如果没有集成 License，播放过程中可能会出现异常。

```
String licenceURL = ""; // 获取到的 licence url
```

```
String licenceKey = ""; // 获取到的 licence key
SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
```

深度定制开发指引

腾讯云播放器 SDK Flutter 插件对原生播放器能力进行了封装， 如果您要进行深度定制开发，建议采用如下方法：

- 基于点播播放，接口类为 `TXVodPlayerController` 或直播播放，接口类为 `TXLivePlayerController` ，进行定制开发，项目中提供了定制开发 Demo，可参考 example 工程里的 `DemoTXVodPlayer` 和 `DemoTXLivePlayer` 。
- 播放器组件 `SuperPlayerController` 对点播和直播进行了封装，同时提供了简单的 UI 交互， 由于此部分代码在 example 目录。如果您有对播放器组件定制化的需求，您可以进行如下操作：

  把播放器组件相关的代码，代码目录： `Flutter/superplayer_widget` ，导入到您的项目中，进行定制化开发。

常见问题

1. iOS 端运行，出现 `No visible @interface for 'TXLivePlayer' declares the selector 'startLivePlay:type:'` 等类似找不到接口错误。
   可以使用如下命令，更新 iOS SDK：

   ```
   rm -rf Pods
   rm -rf Podfile.lock
   pod update
   ```

2. 同时集成 tencent_trtc_cloud 和 Flutter 播放器出现 SDK 或 符号冲突。

   常见异常日志：

   ```
   java.lang.RuntimeException: Duplicate class com.tencent.liteav.TXLiteAVCode found in modules classes.jar
   ```

   此时需要集成flutter 播放器的 Professional 版本，让tencent_trtc_cloud 和 flutter 播放器共同依赖于同一个版的 LiteAVSDK_Professional。注意确保依赖的LiteAVSDK_Professional 的版本必须一样。
   如：依赖 Android 端 TXLiteAVSDK_Professional_10.3.0.11196 和 iOS 端 TXLiteAVSDK_Professional to 10.3.12231 版本，依赖声明如下：

   ```
   tencent_trtc_cloud：2.3.8

   super_player:
     git:
       url: https://github.com/LiteAVSDK/Player_Flutter
       path: Flutter
       ref: release_pro_v1.0.3.11196_12231
   ```

3. 需要同时使用多个播放器实例的时候，频繁切换播放视频，画面呈现模糊。
   在每个播放器组件容器销毁的时候，调用播放器的 `dispose` 方法，将播放器释放。

4. 其余通用 Flutter 依赖问题：
   - 执行 `flutter doctor` 命令检查运行环境，直到出现“No issues found!”。
   - 执行 `flutter pub get` 确保所有依赖的组件都已更新成功。
5. 集成 superPlayer 之后，出现如下 manifest 错误：

```
Attribute application@label value=(super_player_example) from
AndroidManifest.xml:9:9-45
is also present at [com.tencent.liteav:LiteAVSDK_Player:10.8.0.13065]
AndroidManifest.xml:22:9-41 value=(@string/app_name).
Suggestion: add 'tools:replace="android:label"' to <application> element at
AndroidManifest.xml:8:4-51:19 to override.
```

   解决方法：由于播放器 Android SDK 的 AndroidManifest 已经定义过 label，而 flutter 新建项目之后，在 Android 目录的 AndroidManifest 也会定义 label，此处建议根据错误提示，进入您的 Android 项目目录，在 AndroidManifest 的根节点 `manifest` 节点下增加 `xmlns:tools="http://schemas.android.com/tools"`，并在 application 节点下增加 `tools:replace="android:label"`。
6. 集成 superPlayer 之后，出现如下版本错误：

```
uses-sdk:minSdkVersion 16 cannot be smaller than version 19 declared in library
[:super_player]
```

   解决方法：目前播放器 Android SDK 最小支持版本为 android 19，flutter 部分版本默认 Android 最小支持版本为 android 16。建议您将最小支持版本提高到 android 19。具体修改方法为：进入您的 Android 项目的主 module 下，一般为 `app` 目录，将该目录下的 `build.gradle` 中的 `minSdkVersion` 修改为19。
7. 如何提取播放器 SDK 的运行 Log？

   解决方法：播放器 SDK 默认把运行的 log 输出到本地文件，腾讯云技术支持 在帮忙定位问题时，需要这些运行 log 分析问题。Andorid 平台 log 保存在目录：`/sdcard/Android/data/packagename/files/log/tencent/liteav`，iOS 平台 log 保存在目录：`sandbox的Documents/log`。更详细的 log 提取可参考 此教程 。
8. 如何减少控制台 log 输出?

   解决方法：可以通过下面的接口设置 log 输出级别：SuperPlayerPlugin.setLogLevel(TXLogLevel.LOG_LEVEL_NULL)，支持以下 log 级别：

```
class TXLogLevel {
  static const LOG_LEVEL_VERBOSE = 0; // 输出所有级别的log
  static const LOG_LEVEL_DEBUG = 1; // 输出 DEBUG,INFO,WARNING,ERROR 和 FATAL
级别的log
  static const LOG_LEVEL_INFO = 2; // 输出 INFO,WARNNING,ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_WARN = 3; // 输出WARNNING,ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_ERROR = 4; // 输出ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_FATAL = 5; // 只输出FATAL 级别的log
```

```
  static const LOG_LEVEL_NULL = 6; // 不输出任何sdk log
}
```

9. 项目使用过程中，出现原生相关报错，例如

   `错误: 不兼容的类型`、`error: initializing 'BOOL' (aka 'bool') with an expression of incompatible type 'void'`

   等错误，是由于SDK 更新，导致 SDK 与 flutter 端原生代码不兼容。此时只需要更新 SDK 版本即可。

   解决方法：在项目目录下，打开终端，依次输入如下命令：

```
flutter pub cache clean
flutter clean
flutter pub upgrade
flutter pub get
```

   确保命令执行成功，更新本地 flutter 依赖。
   然后在 ios 目录下，打开终端，输入如下命令，更新 IOS 依赖：

```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

   如果问题依然存在，可以尝试删除项目 build 文件夹，并且手动删除您电脑中的 flutter 依赖缓存文件夹 `.pubcache` 。然后重新刷新 flutter pub 依赖再进行编译运行。

10. 安卓点播播放器播放视频，播放器边缘出现平铺拉伸现象。

    该问题是flutter端sdk的纹理渲染问题，可以将flutter版本升级到 flutter 3.7.0以上。

11. flutter调试和测试包运行没问题，但是打正式包会闪退。

    flutter打正式包默认是开启混淆的，播放器SDK需要配置如下混淆规则：

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

12. 播放本地视频无法播放

    flutter播放器支持本地视频播放，需要将正确的本地视频地址传入到视频播放接口中，出现无法播放现象，首先需要检查本地视频地址是否可用，文件是否损坏，如果本地视频没有问题，需要检查应用是否具有存储或者图片/视频读取权限。

13. 运行IOS项目出现 `CocoaPods could not find compatible versions for pod "Flutter"` 等类似报错

    该问题是由于在高flutter开发环境中，已经不再支持IOS低版本，可以检查项目中 Minimum Deployments 配置的IOS版本是否过小，或者是否继承了只支持低IOS版本的依赖。

更多功能

您可以通过运行项目中的 example 体验完整功能，example 运行指引。

播放器 SDK 官网提供了 iOS、Android 和 Web 端的 Demo 体验，请单击这里 。

点播场景

最近更新时间：2024-04-26 11:00:52

阅读对象

本文档部分内容为腾讯云专属能力，使用前请开通 腾讯云 相关服务，未注册用户可注册账号 免费试用 。

通过本文您可以学会

- 如何集成腾讯云视立方 Flutter 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。

基础知识

本文主要介绍视频云 SDK 的点播播放功能，在此之前，先了解如下一些基本知识会大有裨益：

- **直播和点播**
  - 直播（LIVE）的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。
  - 点播（VOD）的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放， 播放中您可以通过进度条控制播放位置，腾讯视频和优酷土豆等视频网站上的视频观看就是典型的点播场景。
- **协议的支持**
  通常使用的点播协议如下，现在比较流行的是 HLS（以“http”打头，以“.m3u8”结尾）的点播地址：

点播协议	优点	缺点
HLS(m3u8)	手机浏览器支持度高	大量小分片的文件组织形式，错误率和维护成本均高于单一文件
MP4	手机浏览器支持度高	格式过于复杂和娇贵，容错性很差，对播放器的要求很高
FLV	格式简单问题少，适合直播转录制场景	手机浏览器支持差，需集成SDK才能播放

特别说明

视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP 和 HLS（m3u8）三种格式的直播地址，以及 MP4、HLS（m3u8）和 FLV 三种格式的点播地址。

SDK 集成

步骤1：集成 SDK 开发包

下载和集成 SDK 开发包，请参考 集成指引 。

步骤2：创建 controller

```
TXVodPlayerController _controller = TXVodPlayerController();
```

步骤3：设置监听事件

```
// 监听视频宽高变化，设置合适的宽高比例,也可自行设置宽高比例，视频纹理也会根据比例进行相应
拉伸
_controller.onPlayerNetStatusBroadcast.listen((event) async {
  double w = (event["VIDEO_WIDTH"]).toDouble();
  double h = (event["VIDEO_HEIGHT"]).toDouble();

  if (w > 0 && h > 0) {
    setState(() {
      _aspectRatio = 1.0 * w / h;
    });
  }
});
```

步骤4：添加布局

```
@override
Widget build(BuildContext context) {
return Container(
  decoration: BoxDecoration(
      image: DecorationImage(
        image: AssetImage("images/ic_new_vod_bg.png"),
        fit: BoxFit.cover,
      )),
  child: Scaffold(
      backgroundColor: Colors.transparent,
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        title: const Text('点播'),
      ),
      body: SafeArea(
          child: Container(
            height: 150,
            color: Colors.black,
            child: Center(
              child: _aspectRatio > 0
                  ? AspectRatio(
                aspectRatio: _aspectRatio,
                child: TXPlayerVideo(controller: _controller),
```

```
          ) : Container(),
        ),
      ))));
}
```

步骤5：播放器初始化

```
// 初始化播放器，分配共享纹理
await _controller.initialize();
```

步骤6：启动播放

通过 url 方式

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

```
// 播放视频资源
String _url =

"http://1400329073.vod2.myqcloud.com/d62d88a7vodtranscq1400329073/59c68fe
75285890800381567412/adp.10.m3u8";
await _controller.startVodPlay(_url);
```

通过 field 方式

```
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
      fileId: "4564972819220421305", psign: "psignxxxxxxx");
await _controller.startVodPlayWithParams(params);
```

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7：结束播放

结束播放时**记得调用 controller 的销毁方法**，尤其是在下次 startVodPlay 之前，否则可能会产生大量的内存泄露以及闪屏问题。

```
@override
void dispose() {
 _controller.dispose();
 super.dispose();
}
```

基础功能使用

1、播放控制

- **开始播放**

```
// 开始播放
_controller.startVodPlay(url)
```

- **暂停播放**

```
// 暂停播放
_controller.pause();
```

- **恢复播放**

```
// 恢复播放
_controller.resume();
```

- **结束播放**

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

- **结束播放器**

```
// 释放controller
_controller.dispose();
```

- **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。

```
double time = 600; // double，单位为 秒
// 调整进度
_controller.seek(time);
```

- **Seek 到视频流指定 PDT 时间点**

  跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。
  注意：播放器高级版 11.7 版本开始支持。

```
int pdtTimeMs = 600; // 单位为 毫秒
_controller.seekToPdtTime(time);
```

- **从指定时间开始播放**

  首次调用 startVodPlay 之前，支持从指定时间开始播放。

```
double startTimeInSecond = 60; // 单位：秒
_controller.setStartTime(startTimeInSecond);  // 设置开始播放时间
_controller.startVodPlay(url);
```

2、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
_controller.setRate(1.2);
```

3、循环播放

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

4、静音设置

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

5 、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。

```
_controller.setAutoPlay(false);  // 设置为非自动播放
_controller.startVodPlay(url);   // startVodPlay 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
// ......
_controller.resume();  // 广告展示完调用 resume 开始播放视频
```

6、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。

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
Map<String, String> httpHeaders = {'Referer': 'Referer Content'};
playConfig.headers = httpHeaders;
_controller.setConfig(playConfig);
```

7、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 **startVodPlay**，否则会产生比较严重的花屏问题。

```
_controller.stopPlay(true);
_controller.enableHardwareDecode(true);
_controller.startVodPlay(url);
```

8、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。可以通过下面方法进行清晰度设置。

```
// 在收到播放器 PLAY_EVT_VOD_PLAY_PREPARED 事件调用 getSupportedBitrates 才会有值返回
List _supportedBitrates = (await _controller.getSupportedBitrates())!; //获取多码率数组
int index = _supportedBitrates[i];  // 指定要播的码率下标
_controller.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 _controller.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。
如果您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，从而避免播放后切换码流。详细方法参见 播放器配置#启播前指定分辨率 。

9、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。可以通过下面方法开启码流自适应。

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

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

10、开启平滑切换码率

在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
/// 设为true，可平滑切换码率，设为false时，可提高多码率地址打开速度
playConfig.smoothSwitchBitrate = true;
_controller.setConfig(playConfig);
```

11、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。
通过 onPlayerEventBroadcast 接口监听播放器事件，进度通知会通过 PLAY_EVT_PLAY_PROGRESS 事件回调到您的应用程序。

```
_controller.onPlayerEventBroadcast.listen((event) async {
  if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {// 更多详细请查看iOS或者Android原生SDK状态码
    // 可播放时长，即加载进度，单位是毫秒
    double playableDuration = event[TXVodPlayEvent.EVT_PLAYABLE_DURATION_MS].toDouble();
    // 播放进度，单位是秒
    int progress = event[TXVodPlayEvent.EVT_PLAY_PROGRESS].toInt();
    // 视频总长，单位是秒
    int duration = event[TXVodPlayEvent.EVT_PLAY_DURATION].toInt();
  }
});
```

12、播放网速监听

通过 onPlayerNetStatusBroadcast 接口监听播放器的网络状态，如： NET_STATUS_NET_SPEED 。

```
_controller.onPlayerNetStatusBroadcast.listen((event) {
   (event[TXVodNetEvent.NET_STATUS_NET_SPEED]).toDouble();
});
```

13、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.

可以通过下面两种方法获取分辨率信息

方法1：通过 onPlayerNetStatusBroadcast **的** NET_STATUS_VIDEO_WIDTH **和** NET_STATUS_VIDEO_HEIGHT **获取视频的宽和高。**

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

```
 _controller.onPlayerNetStatusBroadcast.listen((event) {
  double w = (event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH]).toDouble();
  double h = (event[TXVodNetEvent.NET_STATUS_VIDEO_HEIGHT]).toDouble();
});

// 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
_controller.getWidth();
_controller.getHeight();
```

14、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。

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
playConfig.maxBufferSize = 10; /// 播放时最大缓冲大小。单位：MB
_controller.setConfig(playConfig);
```

15、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。

```
//设置播放引擎的全局缓存目录和缓存大小，//单位MB
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
//设置播放引擎的全局缓存目录
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

16、外挂字幕

注意：此功能需要播放器高级版 11.7 版本开始支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这两种格式的字幕。

最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `VOD_PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。 添加字幕后，并不会主动加载字幕， 调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。选择字幕后，字幕文本内容会通过 `TXVodPlayEvent.EVENT_SUBTITLE_DATA` 。

事件回调，字幕的显示在需要业务方自行处理。

步骤 1：添加外挂字幕

```
// 添加外挂字幕，传入 字幕url， 字幕名称， 字幕类型， 建议在启动播放器前添加
controller.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-
cloud.com/DemoResource/subtitleVTT.vtt", "subtitleName",
TXVodPlayEvent.VOD_PLAY_MIMETYPE_TEXT_SRT)

// 开始播放视频后，监听字幕文本内容回调
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
  // 字幕文本内容，可用于显示
  String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

步骤2：播放后切换字幕

```
// 开始播放视频后，选中添加的外挂字幕， 在收到 VOD_PLAY_EVT_VOD_PLAY_PREPARED 事
件后调用
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.PLAY_EVT_VOD_PLAY_PREPARED) {
    List<TXTrackInfo> trackInfoList = await _vodPlayerController.getSubtitleTrackInfo();
     for (TXTrackInfo tempInfo in trackInfoList) {
       if(tempInfo.name == "subtitleName") {
         // 选中字幕
         _vodPlayerController.selectTrack(tempInfo.trackIndex);
       } else {
         // 其它字幕不需要的话， 进行deselectTrack
```

```
        _vodPlayerController.deselectTrack(tempInfo.trackIndex);
      }
    }
  }
});

// 如果需要，可以监听轨道切换消息
_controller.onPlayerEventBroadcast.listen((event) async {
  if(event["event"] == TXVodPlayEvent.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
    int trackIndex = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_INDEX];
    int errorCode = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_ERROR_CODE];
  }
});
```

步骤3：监听字幕文本内容

```
// 开始播放视频后，监听字幕文本内容回调
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
  // 字幕文本内容，可用于显示
  String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

17、多音轨切换

注意：此功能需要播放器高级版 11.7 版本开始支持。

播放器高级版 SDK 支持切换视频内置的多音轨。用法参见如下代码：

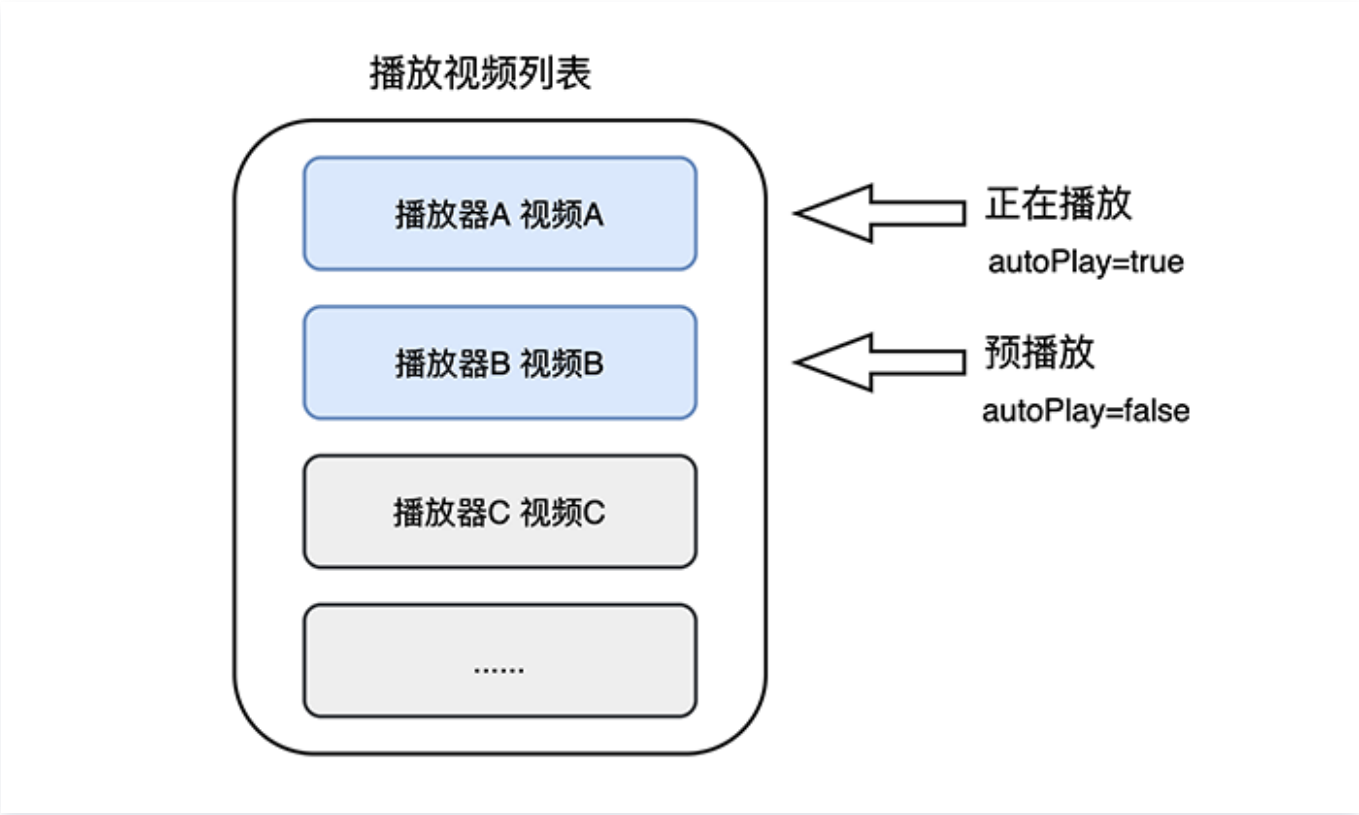
```
// 返回音频轨道信息列表
List<TXTrackInfo> trackInfoList = await _vodPlayerController.getAudioTrackInfo();
for (TXTrackInfo tempInfo in trackInfoList) {
  if(tempInfo.trackIndex == 0) {
    // 通过判断 trackIndex 或者 name 切换到需要的音轨
    _vodPlayerController.selectTrack(tempInfo.trackIndex);
  } else {
    // 不需要的音轨进行 deselectTrack
    _vodPlayerController.deselectTrack(tempInfo.trackIndex);
  }
}
```

进阶功能使用

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 TXVodPlayerController 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 startVodPlay 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";
controller.setAutoPlay(isAutoPlay: true);
controller.startVodPlay(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";
controller.setAutoPlay(isAutoPlay: false);
controller.startVodPlay(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。

```
controller.onPlayerEventBroadcast.listen((event) async {//订阅状态变化
  if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_END) {
    await _controller_A.stop();
    await _controller_B.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。

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.maxBufferSize = 10; // 播放时最大缓冲大小。单位：MB
_controller.setPlayConfig(config); // 把config 传给 controller
```

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 。
使用示例：

通过媒资 URL 预下载

```
//设置播放引擎的全局缓存目录和缓存大小
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
```

```
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路径。
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

String palyrl = "http://****";
//启动预下载
int taskId = await TXVodDownloadController.instance.startPreLoad(palyrl, 3, 1920*1080,
  onCompleteListener:(int taskId,String url) {
    print('taskID=${taskId} ,url=${url}');
  }, onErrorListener: (int taskId, String url, int code, String msg) {
    print('taskID=${taskId} ,url=${url}, code=${code} , msg=${msg}');
  }
);

//取消预下载
TXVodDownloadController.instance.stopPreLoad(taskId);
```

通过媒资 FileId 预下载

```
//设置播放引擎的全局缓存目录和缓存大小
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路径。
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

int retTaskId = -1;
TXVodDownloadController.instance.startPreload(TXPlayInfoParams(appId: 0, fileId: "your fileId"), 1, 720 * 1080,
    onStartListener: (taskId, fileId, url, params) {
      // TXVodDownloadController will call this block for callback taskId and videoInfo
      retTaskId = taskId;
    },
    onCompleteListener: (taskId, url) {
      // preDownload complete
    },
    onErrorListener: (taskId, url, code, msg) {
```

```
    // preDownload error
  });

//取消预下载
TXVodDownloadController.instance.stopPreLoad(retTaskId);
```

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 `TXVodDownloadController` 的视频下载方案实现 HLS 的离线播放。

> **说明:**
> - `TXVodDownloadController` 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

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

```
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路径。
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache 目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

步骤2：开始下载

开始下载有 Fileid 和 URL 两种方式，具体操作如下：

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”。

> **注意:**
> 加密视频只能通过 Fileid 下载，psign 参数必须填写。

```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720, 期望下载此分辨率的流，quality传入 QUALITY_720P)
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
TXVodDownloadMediaInfo mediaInfo = TXVodDownloadMediaInfo();
TXVodDownloadDataSource dataSource = TXVodDownloadDataSource();
dataSource.appId = 1252463788;
dataSource.fileId = "4564972819220421305";
dataSource.quality = DownloadQuality.QUALITY_480P;
dataSource.pSign = "pSignxxxx";
mediaInfo.dataSource = dataSource;
TXVodDownloadController.instance.startDonwload(mediaInfo);
```

URL 方式

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

```
TXVodDownloadMediaInfo mediaInfo = TXVodDownloadMediaInfo();
mediaInfo.url =
"http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq1500005830/00eb06a8860226801143735698 4/video_10_0.m3u8";
TXVodDownloadController.instance.startDonwload(mediaInfo);
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 listener。

```
TXVodDownloadController.instance.setDownloadObserver((event, info) {

}, (errorCode, errorMsg, info) {

});
```

可能收到的任务 event 事件有：

事件	说明
EVENT_DOWNLOAD_START	任务开始，表示 SDK 已经开始下载
EVENT_DOWNLOAD_PROGRESS	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
EVENT_DOWNLOAD_STOP	任务停止，当您调用 `stopDownload` 停止下载，收到此消息表示停止成功
EVENT_DOWNLOAD_FINISH	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

当回调 downlodOnErrorListener 方法的时候，代表下载错误，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。
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 `TXVodDownloadController.instance.stopDownload()` 方法，参数为开始下载传入的 `TXVodDownloadMediaInfo` 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
// 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> downloadInfoList = await
TXVodDownloadController.instance.getDownloadList();
// 获取默认“default”用户已经下载完成的视频
List<TXVodDownloadMediaInfo> defaultUserDownloadList = [];
for(TXVodDownloadMediaInfo mediaInfo in downloadInfoList) {
  if("default" == mediaInfo.userName && mediaInfo.downloadState ==
TXVodPlayEvent.EVENT_DOWNLOAD_FINISH) {
    defaultUserDownloadList.add(mediaInfo);
  }
}
```

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。

```
// 获取某个视频相关下载信息
TXVodDownloadMediaInfo mediaInfo = TXVodDownloadMediaInfo();
mediaInfo.dataSource = TXVodDownloadDataSource();
mediaInfo.dataSource!.appId = 1500005830;
mediaInfo.dataSource!.fileId = "8602268011437356984";
// fileId下载必须传入quality。quality也可以通过
CommonUtils.getDownloadQualityBySize(width, height)来获取
mediaInfo.dataSource!.quality = DownloadQuality.QUALITY_HD;
TXVodDownloadMediaInfo downloadInfo = await
TXVodDownloadController.instance.getDownloadInfo(mediaInfo);
int? duration = downloadInfo.duration;  // 获取总时长
int? playableDuration = downloadInfo.playableDuration; // 获取已下载的可播放时长
double? progress = downloadInfo.progress;  // 获取下载进度
String? playPath = downloadInfo.playPath; // 获取离线播放路径，传给播放器即可离线播放
int? downloadState = downloadInfo.downloadState; // 获取下载状态，具体参考STATE_xxx常
量
```

获取某个 URL 相关下载信息，需要传入 URL 信息。目前url下载不支持嵌套m3u8和mp4下载。

```
TXVodDownloadMediaInfo mediaInfo = TXVodDownloadMediaInfo();
mediaInfo.url =
"http://1253131631.vod2.myqcloud.com/26f327f9vodgzp1253131631/f4bdff7990318682
22924043041/playlist.m3u8";
TXVodDownloadController.instance.getDownloadInfo(mediaInfo);
```

```
// 删除下载信息
bool result = await
TXVodDownloadController.instance.deleteDownloadMediaInfo(mediaInfo);
```

步骤6：播放离线视频

通过以上步骤获取到的mediaInfo的downloadState为EVENT_DOWNLOAD_FINISH，并且mediaInfo的playPath有值，则代表获取视频缓存完成，可以直接传给controller进行播放，代码如下：

```
String cacheVideoUrl = cacheMediaInfo.playPath!;
vodPlayerController.startVodPlay(cacheVideoUrl);
```

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。
在腾讯云控制台提取到appId， 加密视频的fileId 和 psign后，可以通过下面的方式进行播放：

```
// psign 即播放器签名，签名介绍和生成方式参见链接：
https://cloud.tencent.com/document/product/266/42436
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
      fileId: "4564972819220421305", psign: "psignxxxxxxx");
_controller.startVodPlayWithParams(params);
```

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：

```
FTXVodPlayConfig config = FTXVodPlayConfig();
// 如果不配置preferredResolution，则在播放多码率视频的时候优先播放720 * 1280分辨率的码率
config.preferredResolution = 720 * 1280;
config.enableAccurateSeek = true;  // 设置是否精确 seek，默认 true
config.progressInterval = 200;  // 设置进度回调间隔，单位毫秒
config.maxBufferSize = 50;  // 最大预加载大小，单位 MB
_controller.setPlayConfig(config);
```

启播前指定分辨率

播放 HLS 的多码率视频源，如果您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。

```
FTXVodPlayConfig config = FTXVodPlayConfig();
// 传入参数为视频宽和高的乘积(宽 *  高），可以自定义值传入，默认 720 * 1280
config.preferredResolution = 720 * 1280;
_controller.setPlayConfig(config);
```

启播前指定媒资类型

当提前知道播放的媒资类型时，可以通过配置 `FTXVodPlayConfig# mediaType`减少播放器SDK内部播放类型探测，提升启播速度。

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_FILE_VOD;  // 用于提升MP4启播速度
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_HLS_VOD;  // // 用于提升HLS启播速度
_controller.setPlayConfig(config);
```

设置播放进度回调时间间隔

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.progressInterval = 200;  // 设置进度回调间隔，单位毫秒
_controller.setPlayConfig(config);
```

播放器事件监听

您可以通过 `TXVodPlayerController` 的 `onPlayerEventBroadcast` 来监听播放器的播放事件，来向您的应用程序同步信息。

播放事件通知（onPlayerEventBroadcast）

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
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
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：startVodPlay(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
EVT_IMAGESPRIT_WEBVTTURL	雪碧图 web vtt 描述文件下载 URL，10.2版本开始支持
EVT_IMAGESPRIT_IMAGEURL_LIST	雪碧图图片下载 URL，10.2版本开始支持
EVT_DRM_TYPE	加密类型，10.2版本开始支持

通过 onPlayerEventBroadcast 获取视频播放过程信息示例：

```
_controller.onPlayerEventBroadcast.listen((event) async {
  if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_BEGIN || event["event"] ==
TXVodPlayEvent.PLAY_EVT_RCV_FIRST_I_FRAME) {
  // code ...
  } else if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {
  // code ...
  }
});
```

播放状态反馈（onPlayerNetStatusBroadcast）

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，单位 KBps。

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：

```
_controller.onPlayerNetStatusBroadcast.listen((event) async {
  int videoWidth = event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH];
});
```

视频播放状态反馈

视频播放状态会在每一次播放状态切换的时候进行通知。
事件通过枚举类 TXPlayerState 来传递事件

状态	含义
paused	暂停播放
failed	播放失败
buffering	缓冲中
playing	播放中
stopped	停止播放
disposed	控件释放了

通过 onPlayerState 获取视频播放状态示例：

```
_controller.onPlayerState.listen((val) { });
```

系统音量监听

为了方便监控视频播放音量，SDK 在 flutter 层做了对原生层的音量变化通知进行了事件封装，可以直接通过 SuperPlayerPlugin 来监听当前设备的音量变化。
通过 onEventBroadcast 获取设备音量状态示例：

```
SuperPlayerPlugin.instance.onEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

相关事件含义如下：

状态	值	含义
EVENT_VOLUME_CHANGED	1	音量发生变化
EVENT_AUDIO_FOCUS_PAUSE	2	失去音量输出播放焦点，仅使用 Android
EVENT_AUDIO_FOCUS_PLAY	3	获得音量输出焦点，仅使用 Android

画中画事件监听

由于SDK使用的画中画是基于系统提供的画中画能力，进入画中画之后，提供了一系列通知来帮助用户对当前界面做响应的调整。

状态	值	含义
EVENT_PIP_MODE_ALREADY_ENTER	1	已经进入画中画模式
EVENT_PIP_MODE_ALREADY_EXIT	2	已经退出画中画模式
EVENT_PIP_MODE_REQUEST_START	3	开始请求进入画中画模式
EVENT_PIP_MODE_UI_STATE_CHANGED	4	pip UI 状态发生变动，仅在 Android 31以上生效
EVENT_IOS_PIP_MODE_RESTORE_UI	5	重置 UI，仅在 IOS 生效
EVENT_IOS_PIP_MODE_WILL_EXIT	6	将要退出画中画，仅在 IOS 生效

使用 onExtraEventBroadcast 监听画中画事件的示例如下：

```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

直播场景

最近更新时间：2024-02-18 15:20:11

基础知识

本文主要介绍视频云 SDK 的直播播放功能，在此之前，先了解如下一些基本知识会大有裨益：

- **直播和点播**
  直播（LIVE）的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。
  点播（VOD）的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放， 播放中您可以通过进度条控制播放位置，腾讯视频和优酷土豆等视频网站上的视频观看就是典型的点播场景。
- **协议的支持**
  通常使用的直播协议如下，App 端推荐使用 FLV 协议的直播地址（以“http”打头，以“.flv”结尾）：

直播协议	优点	缺点	播放延迟
FLV	成熟度高、高并发无压力	需集成SDK才能播放	2s - 3s
RTMP	优质线路下理论延迟最低	高并发情况下表现不佳	1s - 3s
HLS(m3u8)	手机浏览器支持度高	延迟非常高	10s - 30s

特别说明

- **是否有限制?**
  视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP 和 HLS（m3u8）三种格式的直播地址，以及 MP4、HLS（m3u8）和 FLV 三种格式的点播地址。
- **历史因素**
  SDK 早期版本只有 TXLivePlayer 一个 Class 承载直播和点播功能，但是由于点播功能越做越多，我们最终在 SDK 3.5版本开始，将点播功能单独分离出来，交由 TXVodPlayer 来负责。但是为了保证编译通过，您在 TXLivePlayer 中依然可以看到类似 seek 等点播才具备的功能。

对接攻略

step 1: 创建 Player

视频云 SDK 中的 TXLivePlayer 模块负责实现直播播放功能。对应于 Flutter 是 TXLivePlayerController。

```
TXLivePlayerController controller = TXLivePlayerController();
```

step 2: 渲染 View

接下来我们要给播放器的视频画面找个地方来显示，Flutter 系统中使用 Widget 作为基本的界面渲染单位，所以您只需要准备一个 Widget 并调整好布局就可以了。您可以直接使用 TXPlayerVideo 或者继承它来显示，也可以参考源码实现自定义视图。

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('直播'),
    ),
    body: SafeArea(
      child: Column(
        children: [
          Container(
            height: 150,
            color: Colors.black,
            child: Center(
              child: _aspectRatio>0?AspectRatio(
                aspectRatio: _aspectRatio,
                child: TXPlayerVideo(controller: _controller),
              ):Container(),
            ),
          ),],
      ),
    ),
  );
}
```

step 3: 启动播放

```
String flvUrl =
"http://liteavapp.qcloud.com/live/liteavdemoplayerstreamid_demo1080p.flv";
await _controller.startLivePlay("flvUrl", playType: TXPlayType.LIVE_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：**
> **关于 HLS（m3u8）**
> 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。

step 4: 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
_controller.pause();
// 恢复
_controller.resume();
```

step 5: 结束播放

```
// 停止播放
_controller.stop();
```

step 7: 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，适用场景例如：

- 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：
通过 **onPlayerEventBroadcast** 监听消息，消息编号：**PLAY_EVT_GET_MESSAGE（2012）**。

```
_controller.onPlayerEventBroadcast.listen((event) {//订阅事件分发
  if(event["event"] == 2012) {
     String msg = event["EVT_GET_MSG"];
  }
});
```

step 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
_controller.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。

延时调节

腾讯云 SDK 的直播播放功能，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见下表：

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（例如绝地求生）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如下：

```
// 自动模式
_controller.setLiveMode(TXPlayerLiveMode.Automatic);
// 极速模式
_controller.setLiveMode(TXPlayerLiveMode.Speed);
// 流畅模式
_controller.setLiveMode(TXPlayerLiveMode.Smooth);

// 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?

超低延时播放

支持400ms左右的超低延迟播放时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&txSecret 。
- **播放类型需要指定 ACC**
  在调用 startLivePlay 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 。

功能使用

下文将介绍常见直播播放功能的使用方式。

1、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
_controller.pause();
// 继续
_controller.resume();
```

2、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：

- FTXLivePlayConfig 中的 **enableMessage** 开关置为 **true**。
- TXLivePlayerController 通过 **onPlayerEventBroadcast** 监听消息，消息编号：**PLAY_EVT_GET_MESSAGE（2012）**

```
_controller.onPlayerEventBroadcast.listen((event) {
  if (event["event"] == TXVodPlayEvent.PLAY_EVT_GET_MESSAGE) {
    String msg = event[TXVodPlayEvent.EVT_GET_MSG];
  } else if (event["event"] == TXVodPlayEvent.PLAY_ERR_NET_DISCONNECT) {
    print("网络断开，拉流失败");
  }
});
```

3、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，能直接切到另一条流上。
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
_controller.switchStream("http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab11
e791eae435c87f075e_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。

4、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

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（例如绝地求生）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**

```
FTXLivePlayConfig mPlayConfig = FTXLivePlayConfig();

FTXLivePlayConfig mPlayConfig = FTXLivePlayConfig();
//自动模式
mPlayConfig.autoAdjustCacheTime = true;
mPlayConfig.minAutoAdjustCacheTime = 1;
mPlayConfig.maxAutoAdjustCacheTime = 5;
//极速模式
mPlayConfig.autoAdjustCacheTime = true;
mPlayConfig.minAutoAdjustCacheTime = 1;
mPlayConfig.maxAutoAdjustCacheTime = 1;
//流畅模式
mPlayConfig.autoAdjustCacheTime = false;
mPlayConfig.cacheTime = 5;

_controller.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？

5、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 startLivePlay 函数时，需要指定 playType 为 **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 。

6、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 onPlayerNetStatusBroadcast 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（onPlayerNetStatusBroadcast）。

事件监听

您可以对 TXLivePlayerController 的 onPlayerEventBroadcast 和 onPlayerNetStatusBroadcast 进行监听，之后 SDK 的内部状态信息均会通过 onPlayerEventBroadcast （事件通知）和 onPlayerNetStatusBroadcast （状态反馈）通知给您。

事件通知（onPlayerEventBroadcast）

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
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（PLAY_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（onPlayerNetStatusBroadcast）

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_WIDT	视频分辨率 - 宽

H	
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