ALIBABA CLOUD

阿里云

视频点播
播放器SDK

文档版本：20220330

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认页面，单击确定。**
`Courier字体`	命令或代码。	**执行** `cd /d C:/window` **命令，进入Windows系统文件夹。**
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK简介

阿里云播放器SDK（ApsaraVideo Player SDK）是视频云连接客户端服务的重要一环，本文为您介绍了播放器SDK具备的核心优势以及使用场景等。

简介

阿里云播放器SDK（ApsaraVideo Player SDK，以下简称播放器SDK）是阿里云自研的全端音视频播放工具，为音视频播放提供稳定、流畅、丰富的服务。配合视频点播服务，播放器SDK能够为客户提供云端协同的优异播放体验，以及多场景的解决方案，满足客户的业务需求。播放器SDK具有集成便捷、全端覆盖、播放性能优秀等特点，助力您的业务快速腾飞。

客户价值

为什么选择播放器SDK，请仲冬按照下面的维度提供或补充下内容。
通过集成播放器SDK，您能够更好地连结视频点播服务，打造完善的音视频点播业务闭环。

- 优异性能：播放器SDK稳定支撑海量播放，结合最佳实践综合使用网络优化、预加载、本地缓存等技术实现首屏秒开、极低卡顿率，为用户提供流畅的播放体验。
- 全端覆盖：播放器SDK支持Web、Android、iOS、Flutter、Windows、macOS和Linux等平台，有效支撑多屏播放的业务场景，满足客户多样需求。播放器SDK在不同平台下提供统一的接口设计，显性降低开发者跨平台集成的工作量。
- 消费数据：播放器SDK提供详实的播放数据埋点，可视化的数据看板，帮您把握音视频消费的宏观数据、洞察用户需求。

核心优势

回答怎么样的问题，请仲冬按照下面的维度或其他维度提供或补充下内容。

- 稳定流畅的播放体验
  播放器SDK质量稳定可靠，轻松支撑过亿每日播放vv（video view）、秒开率大于85%、百秒卡顿时长小于100ms。综合播放策略、设备适配软硬解方案确保播放流畅性。
- 全平台覆盖一次性集成
  播放器SDK支持Web、Android、iOS、Flutter、Windows、macOS和Linux等平台，有效支撑多屏播放的业务场景，满足客户多样需求。播放器SDK在不同平台下提供统一的接口设计，显性降低开发者跨平台集成的工作量。
- 丰富多样的播放玩法
  播放器SDK提供DASH伪直播、防遮弹幕、ASS字幕、端侧画质增强等多种功能，为客户提供多场景的解决方案，满足客户的业务需求。
- 复合视频安全方案支持
  视频点播提供标准加密、阿里云视频加密（私有加密）和DRM加密方案，播放器SDK支持上述方案的加密解码，有效解决恶意下载扩散等侵权问题。
- 完善的数据服务体系
  通过播放器SDK埋点，我们提供全盘宏观统计、实时质量监控。针对播放异常，我们提供用户级和播放会话（video view）级的全链路追溯能力，快速定位异常原因，避免负体验影响扩散。
- 直播点播一体服务
  播放器SDK同时支持常见的视频直播、视频点播功能，一次集成多场景使用。

使用场景

- 短视频
  播放器SDK搭配网络优化、预加载、本地缓存等复合最佳实践实现首屏秒开，为您提供流畅的播放体验。
- 长视频
  播放器SDK提供多码率自适应、变速不变调、端侧画质增强等功能，结合云端服务为长视频观影提供进度条缩略图、防遮弹幕、ASS字幕等多样玩法，配合多重安全策略确保视频安全。
- 在线教育
  播放器SDK全端覆盖充分满足您的业务需求，现已支持移动端（Android、iOS）、桌面端（Windows、macOS、Linux）和跨端方案（Web、Flutter）；支持多种加密方案实现有效的版权保护，包含标准加密、阿里云视频加密（私有加密）和DRM加密，您可以按需选择。
- 直播与直播转点播

播放器SDK同时支持视频直播、音视频点播常见协议，无缝对接直播服务，支持直播时移、伪直播、直播录制视频回看，帮助您实现音视频业务闭环。

相关文档

- 如需了解更多播放器SDK功能，请参见 功能详情。
- 如需下载播放器SDK，请参见 SDK下载。
- 如需集成播放器SDK，请根据您的实际业务需要选择播放器SDK类型。详细信息如下：
  - Web播放器快速接入
  - Android播放器快速接入
  - iOS播放器快速接入
  - Flutter播放器快速接入
  - Windows播放器快速接入

2.功能详情

阿里云播放器SDK致力于为集成方提供稳定、流畅、丰富的音视频播放服务。播放器SDK为客户提供多场景的解决方案，满足客户的多样化的业务需求。本文为您介绍了播放器SDK的主要功能，便于您判断播放器SDK是否满足您的需求。请仲冬按照不同的端来划分一下功能，重新提供或补充下内容，下面是以前的内容，可做参考。

功能列表

播放器SDK在各端上都提供丰富的音视频点播、直播功能，主要平台支持的功能列表如下：

> 说明 目前列表只覆盖主流的几个平台，包含Web、Android和iOS。

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
支持协议	音视频格式	支持FLV、HLS、MP4、MP3、RTMP等常见的音视频点播、直播格式。	√	√	√
支持协议	HLS协议	支持HLS、多码率HLS、标准AES-128加密HLS视频流播放。	√	√	√
支持协议	DASH协议	支持标准协议的DASH视频流播放，例如：SegmentBase 、SegmentTemplate。	×	√	√
支持协议	URL播放	支持在线视频、本地视频以URL的方式播放。例如：UrlSource播放。	√	√	√
支持协议	Vid播放	支持Vid方式播放，同时对视频下发多个清晰度的播放地址（若有）。例如：VidAuth播放、VidSts播放。	√	√	√
支持协议	完整UI	SDK包含完整UI，集成方可以根据自身需求选用。	√	√	√
基础功能-播放控制	基础控制	支持开始、结束、暂停、seek、自动播放等播放控制功能。	√	√	√

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
基础功能-播放控制	seek	支持拖动到指定位置（UI支持手势）；支持已经缓冲的视频内容在拖动时不清除缓冲内容并快速拖动。	√	√	√
基础功能-播放控制	精确seek	支持精确到帧级别拖动到指定位置。	√	√	√
基础功能-播放控制	续播	支持设置续播起播时间点。	√	√	√
基础功能-播放控制	循环播放	支持音视频播放结束后自动重播。	√	√	√
基础功能-播放控制	列表播放	支持（短视频）列表播放功能，提升加载速度。	√	√	√
基础功能-播放控制	倍速播放	支持变速播放，支持音频变速不变调。	√ 说明 支持的倍速范围为0.5~2倍。	√ 说明 支持的倍速范围为0.5~5倍。	√ 说明 支持的倍速范围为0.5~5倍。
基础功能-播放控制	清晰度（手动）切换	支持视频点播的多路清晰度流切换。	√	√	√
基础功能-播放控制	截图	支持截取当前播放画面帧并保存为静态图片。	部分支持 说明 受平台、播放模式的影响略有差异。更多信息，请参见开启截图功能。	√	√
基础功能-播放控制	渲染数据输出	支持边渲染边输出PCM和YUV数据。	×	√	√
基础功能-播放控制	事件回调	支持对播放状态回调、首帧回调、播放完成或失败回调。	√	√	√

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
基础功能-显示效果	填充	支持画面填充和画面裁剪两种填充模式。	√	√	√
基础功能-显示效果	旋转	支持0°、90°、180°和270°四个视频画面渲染角度设置。	√	√	√
基础功能-显示效果	镜像	支持无镜像、水平镜像和垂直镜像三种镜像模式设置。	√	√	√
基础功能-显示效果	亮度调节	支持系统的亮度调节（UI支持手势）。	√	√	√
基础功能-显示效果	自定义播放器尺寸	支持自定义设置播放器的宽高。	√	√	√
基础功能-显示效果	HDR视频播放	支持HDR10/HLG等多种HDR格式；支持根据机型画像精确判断是否支持HDR及规格实现精确选流。	√	√	√
基础功能-显示效果	画中画（小窗）播放	支持切换到画中画以小窗形式播放。	√	√	部分支持 说明 不支持在集成的App外画中画播放。
基础功能-音量效果	音量调节	支持播放视频时调节系统音量；支持软件音量0~2倍放大。	√	√	√
基础功能-音量效果	静音	支持开启和关闭静音功能。	√	√	√
基础功能-音量效果	纯音频播放	支持只播放视频文件中的音频和常见的音频文件，例如：MP3、AAC。	√	√	√
基础功能-音量效果	后台播放	支持界面切到后台后继续播放音频。	√	√	√

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
进阶功能-播放性能	播放失败重试	播放失败时自动重试。	√	√	√
进阶功能-播放性能	HTTP 2.0	支持HTTP 2.0协议。	部分支持 说明 取决于浏览器是否支持。更多信息，请参见HTTP 2。	√	√
进阶功能-播放性能	HTTPDNS	支持HTTPDNS，实现域名防劫持、精准调度、实时解析生效。	×	√	√
进阶功能-播放性能	预加载	/	√	√	√
进阶功能-播放性能	本地缓存	/	×	√	√
进阶功能-播放性能	软硬解切换	支持H.264和H.265的硬解码、软解码功能，并支持切换。	部分支持 说明 Web播放器SDK不支持H.265。	√	√
进阶功能-播放性能	解码策略黑名单	支持设置硬解码黑名单。	√	√	√
进阶功能-播放性能	网络自适应播放多码率视频流	支持多码率HLS、DASH的无缝切换。	√	√	√
进阶功能-播放性能	网速显示	支持播放器实例的实时网速上报。	√	√	√
进阶功能-播放性能	边播边缓存	支持视频播放的同时缓存下载后面的内容；重复播放视频时达到省流效果。	√	√	√

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
进阶功能-互动功能	字幕	支持导入自定义的字幕文件。	√ 说明 目前仅支持WebVTT字幕。	√ 说明 支持SRT、SSA、ASS字幕。	√ 说明 支持SRT、SSA、ASS字幕。
进阶功能-互动功能	弹幕	支持弹幕、防遮弹幕。	√	√	√
进阶功能-视频安全	HLS标准加密	支持标准AES-128加密方案。	√	√	√
进阶功能-视频安全	阿里云加密（私有加密）	支持阿里云加密（私有加密）方案，防止视频泄露和盗链问题。	部分支持 说明 Web播放器SDK在iOS平台不支持私有加密。	√	√
进阶功能-视频安全	安全下载	支持通过唯一应用下载视频并进行加密。	部分支持 说明 Web播放器SDK在iOS平台不支持安全下载。	√	√
进阶功能-视频安全	DRM加密	支持Widevine、Fairplay等DRM加密方案。	√	√	√
数据服务	日志上报	支持上报播放器SDK日志，统计音视频点播、直播相关播放埋点信息。	√	√	√
数据服务	播放数据大盘	支持观测播放量、播放用户数等宏观数据。	√	√	√
数据服务	播放异常追溯	支持按照指定用户、播放会话对播放历史进行追溯，定位播放异常原因。	×	√	√

分类	功能点	功能说明	Web播放器SDK	Android播放器SDK	iOS播放器SDK
直播功能	低延时直播	支持LHLS、DASH的低延时流播放。	√	√	√
直播功能	超低延时直播	支持阿里云RTS超低延时播放。	√	√	√
直播功能	自动重连	支持直播的自动重连功能。	√	√	√
直播功能	动态追帧	支持直播的动态追帧，降低延时。	√	√	√

3.SDK下载

本文提供播放器SDK下载及Demo体验指引。

SDK下载

- 最新版本（推荐）：播放器SDK最新版本及其下载地址请参见 播放器SDK下载 。
- 历史版本：播放器SDK历史版本的更新记录及其下载地址请参见 播放器SDK发布历史 。

Demo体验

阿里云播放器SDK Demo提供完整的交互界面和业务源码，集成方可以通过Demo体验播放器SDK的功能。播放器SDK Demo为开发者提供最佳实践参考，帮助开发者快速实现业务需求，节约开发时间和成本。详细信息，请参见Demo体验。

4.Web播放器

4.1. Web播放器简介

阿里云Web播放器SDK支持HTML5（以下简称H5）和Flash两种播放模式。由于Flash Player已停止服务，主流浏览器均不支持Flash播放。在Internet Explorer及其他不支持H5播放的浏览器下使用需要切换至Flash模式。本文简要介绍Web播放器两种播放模式的功能支持和浏览器适配情况。

音视频格式支持

Web播放器SDK支持的音视频协议及编码格式如下。

> **注意** 由于RTMP视频格式依赖支持Flash的早期浏览器，RTMP视频不能通过H5模式播放。H5模式可使用FLV播放。

播放模式	视频编码格式	视频协议	音频编码格式	音频文件格式
H5	H.264	MP4、FLV、HLS	AAC	MP3
Flash（已停止更新）	H.264	MP4、FLV、HLS、RTMP	AAC、MP3	MP3

H5浏览器适配说明

H5模式同时支持桌面端和移动端浏览器环境。

桌面端浏览器适配

> **说明** 桌面端浏览器播放FLV、HLS视频时必须启用 **跨域访问**。

浏览器	MP4	FLV	HLS	MP3
Chrome	✓	34及以上版本	34及以上版本	✓
Firefox	✓	49及以上版本	49及以上版本	✓
IE	IE 9及以上版本	• 点播视频：Windows 8.1及以上、IE 11及以上版本 • 直播视频：Windows 8.1及以上、IE 11以上版本 **注意** 由于*flv.js*在IE 11浏览器下的 `mseLiveFlvPlayback` 属性为false，FLV的直播流在Windows 8及以上系统的IE 11浏览器下无法播放。	Windows 8.1及以上、IE 11及以上版本	IE 9及以上版本
Edge	✓	✓	✓	✓

浏览器	MP4	FLV	HLS	MP3
Opera	✓	✓	✓	✓
Safari	✓	8及以上版本	8及以上版本	✓

移动端浏览器适配

> **说明** Web播放器H5模式在移动端不支持播放FLV视频。

Android端H5浏览器适配

浏览器	MP4	FLV	HLS	MP3
Chrome	✓	×	34及以上版本	✓
Firefox	✓	×	49及以上版本	✓
微信	✓	×	✓	✓
Edge	✓	×	✓	✓
Opera	✓	×	✓	✓

iOS端H5浏览器适配

浏览器	MP4	FLV	HLS	MP3
Chrome	✓	×	34及以上版本	✓
Firefox	✓	×	49及以上版本	✓
微信	✓	×	✓	✓
Edge	✓	×	✓	✓
Opera	✓	×	✓	✓
Safari	✓	×	8及以上版本	✓

H5功能适配说明

功能	桌面端	移动端	如何设置
全屏播放	默认非全屏播放。	视频默认全屏播放，针对不同的情况需要不同的设置。 **注意** iOS 10以下版本的Safari浏览器无法禁止视频自动全屏播放。	进阶功能

功能	桌面端	移动端	如何设置
自动播放	限制来源于浏览器自身而不是Web播放器SDK。 • macOS High Sierra Safari 11及以上版本限制自动播放。 • Chrome 55及以上版本限制自动播放。	默认禁止。开启需要特殊设置。 说明 不排除部分浏览器和WebView允许自动播放，Android系统中较为常见。	进阶功能
音量调节	支持	由于 `video.volume` 在iOS 和一些Android系统中是可读属性，阿里云Web播放器提供的音量调节方法 `getVolume` 和 `setVolume` 在iOS系统和部分Android系统会失效。	基础功能
倍速播放	支持	部分移动端浏览器不支持设置倍速，比如Android系统的微信浏览器。	基础功能
HLS标准加密视频播放	• Chrome • FireFox • Safari • Edge • Windows 8.1及以上、IE 11及以上版本	支持	视频加密播放
阿里云私有加密视频播放 说明 出于安全考虑，阿里云私有加密不支持用真实的加密视频调试。 2022.03.22新增 note	• Chrome（推荐） • FireFox • Safari • Edge • Windows 8.1及以上、IE 11及以上版本 说明 部分第三方浏览器会强制劫持播放器，使得阿里云私有加密失效，进而导致无法播放。此时建议采用HLS标准加密或DRM加密方案。	iOS平台不支持。 Android平台仅Chrome for Android支持。	

功能	桌面端	移动端	如何设置
视频直播DRM加密视频播放	Windows系统 • Chrome • Opera • FireFox • Edge macOS系统 • Chrome • Safari • FireFox • Opera • Edge	iOS平台 • Chrome • Safari Android平台 • Android 10及以上的Chrome • Edge	
点播DRM加密视频播放	Windows系统 • Chrome • Opera • FireFox • Edge macOS系统 • Chrome • Safari • FireFox • Opera • Edge	iOS平台 • Chrome • Safari Android平台 • Android 10及以上的Chrome • Edge	

Flash浏览器适配说明

Flash模式仅支持桌面端浏览器环境。具体的浏览器支持如下：

> 说明 在IE 8浏览器使用Flash模式播放时，需要在页面添加 *json.min.js*的引用。详情请参见快速接入。

浏览器	MP4	FLV	HLS	RTMP	MP3
Chrome	✓	✓	✓	✓	✓
Firefox	✓	✓	✓	✓	✓
IE	IE 8及以上版本	IE 8及以上版本	IE 8及以上版本	IE 8及以上版本	IE 8及以上版本
Edge	✓	✓	✓	✓	✓
Opera	✓	✓	✓	✓	✓
Safari	✓	✓	✓	✓	✓

4.2. 在线体验及Demo源码

本文介绍基于阿里云Web播放器SDK的在线体验方式，并提供Demo源码。

桌面端Demo体验

Web播放器SDK提供可视化的在线体验。您可通过 在线配置 访问。在线配置提供基础的播放配置、样式配置，支持生成HTML5、Flash两套代码。

> **说明** H5模式下，当采用直播的地址播放方式时，播放地址仅支持FLV格式，不支持RTMP格式。

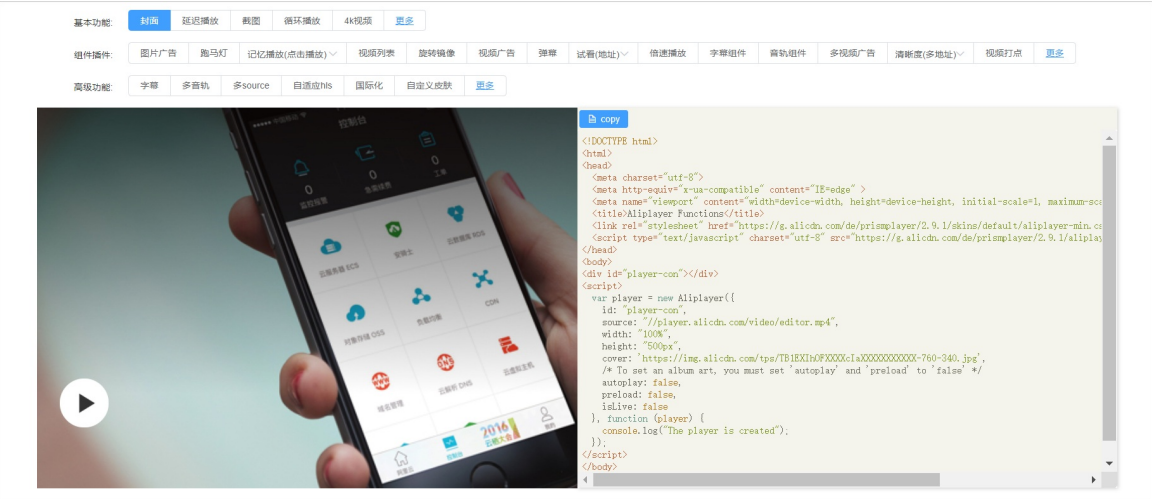
移动端Demo体验

使用钉钉APP，扫描以下二维码体验Web播放器SDK移动端Demo。

> **注意** Android手机上微信、QQ等浏览器存在劫持播放器的情况下，有些功能会失效。

功能Demo源码

Web播放器提供在线功能展示并在功能展示页面同步展现代码实现。详细的功能列表（基础功能、组件及高级功能）及示例代码请参见功能展示。



Vue Demo源码

2022.01.11：根据用户反馈新增该section
提供基于Vue的播放器Demo源码，方便开发者使用。

Demo地址请参见阿里云播放器 Vue Demo。

移到最佳实践或者SDK合集 微信小程序

微信小程序缺少相关的DOM API和BOM API，这一区别导致了前端开发非常熟悉的一些库，例如 jQuery、Zepto 等，在微信小程序里不能运行。 同理Web播放器SDK也是基于浏览器环境的，在微信小程序里不能运行，因此，需要使用小程序自带的Video组件去播放视频，详情请参见视频点播微信小程序Demo。

4.3. 快速接入

本文介绍如何快速接入Web播放器并完成最基础的视频播放。

接入前须知

阿里云Web播放器SDK支持HTML5和Flash两种播放模式，请您提前明确所需集成的播放器模式，并了解相关功能支持和浏览器适配情况。关于功能支持和浏览器的适配说明，详细内容请参见Web播放器简介。

快速接入

1. 引入js文件。

   Web播放器不依赖于任何的前端js库，只需要在页面中引用js文件，就可以进行初始化。

   ```
   <head> <link rel="stylesheet"
   href="https://g.alicdn.com/de/prismplayer/2.9.19/skins/default/aliplayer-min.css" /
   > //（可选）如果您的使用场景需要用到H5播放器，则需引用此css文件。 <script charset="utf-8"
   type="text/javascript" src="https://g.alicdn.com/de/prismplayer/2.9.19/aliplayer-mi
   n.js"></script> //（必须）引入js文件。 </head>
   ```

   引用的js文件同时包含了Flash和H5跨终端自适应的逻辑。如果您只是想使用其中一种播放技术，也可以只引用对应技术的js文件，从而获得更小的文件体积：

   - H5模式，示例如下：

     ```
     https://g.alicdn.com/de/prismplayer/2.9.19/aliplayer-h5-min.js
     ```

   - Flash模式，示例如下：

     ```
     https://g.alicdn.com/de/prismplayer/2.9.19/aliplayer-flash-min.js
     ```

在IE 8浏览器使用Flash播放器时，需要在页面添加json.min.js的引用。示例如下：

```
https://g.alicdn.com/de/prismplayer/

          2.9.19
        /json/json.min.js
```

2. 提供挂载元素。

```
<body> <div id="J_prismPlayer"></div> </body>
```

3. 实例化播放器。
   - Web播放器SDK支持5种点播播放方式，包括：URL播放、Vid+PlayAuth播放（推荐）、 STS播放、MPS播放、加密播放。各播放方式的代码示例请参见点播视频播放。
   - Web播放器SDK支持2种直播播放方式，URL播放和加密播放。各播放方式的代码示例请参见 直播视频播放。

点播视频播放

中国站

点播URL播放 | VID+PlayAuth播放 | 点播STS播放 | MPS播放 | 点播加密播放

直播视频播放

中国站

直播URL播放 | 直播DRM加密播放

相关文档

- 接口说明
- 基础功能
- 进阶功能

4.4. 基础功能

在使用Web播放器的过程中，常用的功能可能由于播放模式、播放方式或浏览器环境的不同而需要不同的设置。本文提供Web播放器基础功能的使用示例。

常用设置

H5模式和Flash模式播放FLV和HLS视频需要配置跨域访问。配置方法请参见 配置跨域访问。

- H5模式如未配置跨域访问，播放时浏览器会报以下错误：
  ```
  No 'Access-Control-Allow-Origin' header is present on the requested resource.
  ```
- Flash模式如未配置跨域访问，播放时浏览器会报以下错误：

Android系统微信或QQ浏览器上播放视频时，由于腾讯X5浏览器会挟持视频自动全屏播放，具体操作，请参见如何启用H5的同层播放。

接口调用规则

Web播放器的基础功能由属性或接口实现。其中，接口的调用规则如下：

> 说明 属性和接口的详细描述请参见 接口说明。

- HTML5播放模式
  在创建播放器构造函数的回调函数里调用。示例如下：

```
//H5播放模式
 var player = new Aliplayer({},function(player) {
    player.play();
 });
```

- Flash播放模式
  在ready事件发生之后或创建播放器ready回调里调用。示例如下：

```
//Flash播放模式
 player.on('ready',function(e) {
    player.play();
 });
```

控制播放

Web播放器支持从指定时间点播放和暂停播放等操作。H5模式和Flash模式使用的接口一样。 stop接口有BUG，暂时不对外透出。指定时间播放即seek。

从指定时间开始播放

指跳转到某个时刻进行播放，由 `seek` 接口实现。示例如下：

```
//time为指定的时间，单位：秒。
player.seek(time)
```

暂停播放

指暂停播放视频，由 `pause` 接口实现。示例如下：

```
player.pause()
```

设置显示模式

Web播放器仅H5模式支持旋转、镜像等显示设置。

> 说明
> - 支持桌面端和iOS的浏览器。
> - 支持Android（Chrome浏览器、Firefox浏览器）。
> - 微信和大部分浏览器，由于视频被劫持播放，使用自带的播放器，所以不支持此功能。

旋转

指画面按指定角度旋转，由 `setRotate` 接口实现。设置后还可查询旋转角度。示例如下：

```
//设置旋转角度。<角度>正数为顺时针旋转，负数为逆时针旋转。如：player.setRotate(180)表示顺时针旋转180度。
player.setRotate(<角度>)
//获取旋转角度。
player.getRotate()
```

镜像

支持水平镜像和垂直镜像，由 `setImage` 接口实现。示例如下：

```
//水平镜像
player.setImage('horizon')
//垂直镜像
player.setImage('vertical')
```

此外，Web播放器还提供两个属性 `videoHeight` 和 `videoWidth` 用于设置视频的宽度和高度，高度和宽度一般比容器的小，这样旋转和镜像时不会溢出到父容器外面，示例如下：

```
 width: '100%', //容器的大小
 height: '100%', //容器的大小
 videoHeight:"200px", //视频的高度大小
```

获取播放信息

缓存暂时不提供，Quote真岛：播放器不是没有缓存视频的功能，播放器的缓存功能是服务于播放流程的，不是一个开放给用户的产品化功能Web播放器支持获取当前的播放进度和播放时长信息。

获取当前播放进度

指获取当前的播放时刻，由 `getCurrentTime` 接口实现。示例如下：

```
//接口返回的时间单位为秒。
player.getCurrentTime()
```

获取播放时长

指获取视频总时长。需要在视频加载完成以后才可以获取到，可以在play事件后获取。由 `getDuration` 实现，示例如下：

```
player.getDuration()
```

监听播放状态

复用源在Web播放器的接口说明 指监听播放器的状态。由 `getStatus` 接口实现。返回值包括：

- **init**：初始化。
- **ready**：准备。
- **loading**：加载中。
- **play**：播放。
- **pause**：暂停。
- **playing**：正在播放。
- **waiting**：等待缓冲。
- **error**：错误。
- **ended**：结束。

示例如下：

```
player.getStatus()
```

设置音量

设置音量包括音量调节和静音设置。注意的复用源在Web播放器简介。

> **注意**
> 由于 `video.volume` 在iOS 和一些Android系统中是可读属性，阿里云Web播放器提供的音量调节方法 `getVolume` 和 `setVolume` 在iOS系统和部分Android系统会失效。

音量调节

指调节音量大小，由 `setVolume` 接口实现。设置后还可获取音量信息。示例如下：

```
//volume的值为0~1之间的实数。
player.setVolume(0)
//获取音量信息。
player.getVolume()
```

静音设置

指将播放中的视频设置为静音状态，由 `mute` 接口实现。示例如下：

```
player.mute()
```

倍速播放

仅H5模式支持倍速播放。Web播放器SDK默认的UI自带倍速功能，用户观看时可通过界面选择倍速。如果自定义UI，可通过 `setSpeed` 接口实现倍速功能。

> **说明** 关闭倍速设置请参见 **接口说明** 中关于 `setSpeed` 的描述。

示例如下：

```
//设置倍速。以下示例表示设置为2倍速。
player.setSpeed(2)
```

多清晰度播放

指通过设置多路清晰度流的地址，达成多清晰度播放的效果。

- 如果使用VID+PlayAuth方式播放，无需额外设置。Web播放器SDK会从点播服务获取清晰度列表。观众点击播放界面控制栏里的设置按钮可以看到清晰度列表。

  > **说明** VID+PlayAuth方式下，H5播放模式可以通过设置format属性选择播放MP4或MP3播放格式，默认为MP4格式播放。

- 如果使用URL播放方式播放（即通过设置source属性播放），需要在source属性设置中通过JSON结构的键值对（Key-Value Pair）指定多路清晰度流的地址。设置生效后观众点击播放界面控制栏里的设置按钮可以看到清晰度列表。

> **说明**
> 如需改变清晰度列表的UI，可以通过引用清晰度组件实现。代码示例请参见 **功能展示**。

URL播放方式下，source属性设置中可通过JSON结构的键值对（Key-Value Pair）包括：

```
"OD": "<原画URL>"
"FD": "<流畅URL>"
"LD": "<标清URL>"
"SD": "<高清URL>"
"HD": "<超清URL>"
"2K": "<2K URL>"
"4K": "<4K URL>"
```

以下是URL播放方式设置清晰度的示例代码：

```
//示例设置清晰度为超清和高清的地址。
source:'{"HD":"http://******/player/hdexample.mp4","SD":"http://******/player/sdexample.m
"}'
```

Web播放器还支持通过设置qualitySort属性，表示启用升序还是降序排列清晰度。

- desc表示按倒序排序（从大到小排序）。
- asc表示按正序排序（从小到大排序）。

> 说明
> - 清晰度切换会记住用户当前选择的清晰度，下次重新打开播放视频时，会优先选择上次选择的清晰度，没有则按默认逻辑选择低清晰度播放。
> - 用户选择的清晰度不能播放器时，会自动切换到下一个清晰度并提示，仅H5支持。

循环播放

Web播放器支持通过设置 `rePlay` 属性或监听 `ended` 事件实现循环播放。

设置rePlay属性示例

```
rePlay:true
```

监听ended事件示例

```
player.on('ended',function(){
   player.replay()
})
```

相关文档梳理后重新提供链接

- 进阶功能
- 插件组件
- 接口说明
- 常见问题

4.5. 进阶功能

本文介绍Web播放器SDK提供的进阶功能，内容涵盖常见的播放控制功能和适用于长视频场景功能的集成使用。

播放控制

自动播放

Web播放器SDK支持自动播放，但由于浏览器自身的限制，在Web播放器SDK中无法通过设置 `autoplay` 属性或者调用 `play()` 方法实现自动播放。只有视频静音才可以实现自动播放或者通过用户行为手动触发播放（例如：初始化后，手动调用 `setVolume` 方法对视频进行静音处理）。

> **说明**
> - 桌面端浏览器有以下限制：
>   - Safari浏览器：macOS High Sierra Safari 11及以上版本限制自动播放。
>   - Chrome浏览器：Chrome 55及以上版本限制自动播放。
> - 移动端浏览器不排除部分浏览器和WebView允许自动播放，Android系统中较为常见。

以下示例以H5模式下的URL播放方式为例展示如何手动设置视频静音：

```
//初始化后，手动对视频进行静音处理
var player = new Aliplayer({
"id": "player-con",
//您的播放地址。示例：example.aliyundoc.com/video/****.mp4
"source":"<your URL>",
"width": "100%",
"height": "500px",
"autoplay": true,
"preload": true,
"controlBarVisibility": "hover",
"useH5Prism": true
}, function (player) {
player.mute()
//[或者] player.setVolume(0)
}
);
```

连续播放

连续播放是指当前视频播放完毕时自动连续播放下一个视频。连续播放实现受播放方式，播放器模式和播放场景影响。

- URL播放方式
  H5模式和Flash模式的行为都是一致的，需要订阅 `ended` 事件，在 `ended` 事件里，调用 `loadByUrl` 方法，参数为下一个视频的地址。示例如下：

  ```
  function endedHandle()
  {
    var newUrl = "";
    player.loadByUrl(newUrl);
  }
  player.on("ended", endedHandle);
  ```

- Vid+PlayAuth播放方式

- H5模式在 `ended` 事件里调用 `replayByVidAndPlayAuth` 方法，参数为 `vid` 和新的 `playauth` 值。示例如下：

```
function endedHandle()
{
 var newPlayAuth = "";
 player.replayByVidAndPlayAuth(vid,newPlayAuth);
}
player.on("ended", endedHandle);
```

- Flash模式没有提供切换 `vid` 和 `playauth` 的方法，需要销毁重新创建播放器。示例如下：

```
function endedHandle()
{
   var newPlayAuth = "";
   player.dispose(); //销毁
   $('#J_prismPlayer').empty();//id为html里指定的播放器的容器id
    //重新创建
   player = new Aliplayer({
             id: 'J_prismPlayer',
             autoplay: true,
             playsinline:true,
             vid: vid,
             playauth:newPlayAuth,
             useFlashPrism:true
         });
   }
}
player.on("ended", endedHandle);
```

> 注意 `playauth` 的默认有效期只有100s，调用 `replayByVidAndPlayAuth` 方法时，需要重新获取音视频播放凭证 。

- 地址协议不一样的切换处理
  如果原来播放的是mp4格式的视频，现在新的地址是hls的视频地址，这种情况只能重新创建播放器。示例如下：

```
function endedHandle()
{
    var newUrl = ""; //新的播放地址
    player.dispose(); //销毁
     //重新创建
   setTimeout(function(){
     player = new Aliplayer({
              id: 'J_prismPlayer',
              autoplay: true,
              playsinline:true,
              source:newUrl
          });
      }
   },1000);
}
player.on("ended", endedHandle);
```

自定义播放器外观和控件 开发输入

Web播放器SDK支持自定义播放器外观（播放器皮肤）、控件（播放器控制栏UI、报错UI等）是否显示以及显示的位置。

- 播放器控制栏UI
  通过修改skinLayout属性定制组件是否显示及显示位置。更多信息，请参见 配置skinLayout属性 。
  - 视频点播H5模式默认配置

```
skinLayout:[
   {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {name: "H5Loading", align: "cc"},
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay"},
    {name:"tooltip", align:"blabs",x: 0, y: 56},
    {name: "thumbnail"},
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
        {name: "progress", align: "blabs", x: 0, y: 44},
        {name: "playButton", align: "tl", x: 15, y: 12},
        {name: "timeDisplay", align: "tl", x: 10, y: 7},
        {name: "fullScreenButton", align: "tr", x: 10, y: 12},
        {name:"subtitle", align:"tr",x:15, y:12},
        {name:"setting", align:"tr",x:15, y:12},
        {name: "volume", align: "tr", x: 5, y: 10}
      ]
    }
  ]
```

- 视频点播Flash模式默认配置

```
skinLayout:[
    {name:"bigPlayButton", align:"blabs", x:30, y:80},
    {
      name:"controlBar", align:"blabs", x:0, y:0,
      children: [
        {name:"progress", align:"tlabs", x: 0, y:0},
        {name:"playButton", align:"tl", x:15, y:26},
        {name:"nextButton", align:"tl", x:10, y:26},
        {name:"timeDisplay", align:"tl", x:10, y:24},
        {name:"fullScreenButton", align:"tr", x:10, y:25},
        {name:"streamButton", align:"tr", x:10, y:23},
        {name:"volume", align:"tr", x:10, y:25}
      ]
    },
    {
      name:"fullControlBar", align:"tlabs", x:0, y:0,
      children: [
        {name:"fullTitle", align:"tl", x:25, y:6},
        {name:"fullNormalScreenButton", align:"tr", x:24, y:13},
        {name:"fullTimeDisplay", align:"tr", x:10, y:12},
        {name:"fullZoom", align:"cc"}
      ]
    }
]
```

- 报错UI
  Web播放器SDK提供默认的报错UI，同时提供两种自定义错误UI的方式。详细信息，请参见 H5自定义错误UI。
  - 通过CSS自定义皮肤
    保留播放器原有的布局和显示，通过重写CSS自定义皮肤，修改背景颜色、是否显示、字体、位置等。
  - 重写UI
    重写UI需要订阅错误事件。

- Web播放器SDK皮肤
  Web播放器SDK UI不能满足业务需求时，可以通过配置播放器CSS样式来具体指定播放器皮肤。
  H5模式播放器皮肤设置

> 说明 具体可以参照播放器的CSS配置文件 *aliplayer-min.css*。以下代码以大播放按钮为例，更多信息请参见设置播放器皮肤。

```
.prism-player .prism-big-play-btn {
  width: 90px;
  height: 90px;
  background: url("//gw.alicdn.com/tps/TB1YuE3KFXXXXaAXFXXXXXXXXXX-256-512.png") no-r
epeat -2px -2px;
}
```

Flash模式播放器皮肤设置
在 `http://[domain]/[path]/` 目录下存放*skin.png*与*skin.xml*文件，并在player的输入参数中添加 `skinRes: "http://domain/path/skin"` 。

> **说明** 以下代码以大播放按钮为例，更多信息请参见 **设置播放器皮肤** 。

```
<SubTexture name="bigPlayDown" x="2" y="94" width="90" height="90"/>
<SubTexture name="bigPlayOver" x="94" y="2" width="90" height="90"/>
<SubTexture name="bigPlayUp" x="2" y="2" width="90" height="90"/>
```

自定义视频封面图 开发输入

点播上传的每一个视频都设置了封面图片，并提供了多种设置和修改视频封面的方法。视频上传前可以选择指定的图片作为封面，或进行视频截图并选择一张作为封面。视频上传完成后也可以对封面进行更新。设置封面的方法如下：

- 通过点播控制台设置封面。具体操作，请参见 **设置视频封面** 。
- 通过播放器属性cover设置封面。

```
var player = new Aliplayer({
 "id": "player-con",
 "source":"//player.alicdn.com/video/aliyunm****.mp4",
  "cover":"封面地址",
},
 function () { }
);
```

视频截图

Web播放器SDK 2.1.0以上的版本支持在视频播放过程中截图。截取的图片为 `image` 或 `jpeg` 类型。截图功能需要单独开启，截图返回数据为当前播放时间、base64和二进制的图片。

开启截图功能

- H5模式下开启截图功能

> **注意** FLV视频在Safari浏览器下不支持截图功能。即使启用截图按钮也不会出现。此外，H5模式下的截图是通过Canvas实现的，播放域名需添加允许跨域访问的Header，详情请参见**配置跨域访问** 。

在skinLayout数组里添加snapshot UI。示例如下：

```
    skinLayout:[
    {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {
      name: "H5Loading", align: "cc"
    },
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay"},
    {name:"tooltip", align:"blabs",x: 0, y: 56},
    {name: "thumbnail"},
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
        {name: "progress", align: "blabs", x: 0, y: 44},
        {name: "playButton", align: "tl", x: 15, y: 12},
        {name: "timeDisplay", align: "tl", x: 10, y: 7},
        {name: "fullScreenButton", align: "tr", x: 10, y: 12},
        {name:"subtitle", align:"tr",x:15, y:12},
        {name:"setting", align:"tr",x:15, y:12},
        {name: "volume", align: "tr", x: 15, y: 10},
        {name: "snapshot", align: "tr", x: 5, y: 12},
      ]
    }
  ]
```

H5模式下截图需要设置video的允许匿名跨域访问属性。示例如下：

```
  extraInfo:{
    crossOrigin:"anonymous"
  }
```

- Flash模式下开启截图功能

> **注意** RTMP流不支持截图。

Flash模式下通过snapshot属性开启，设置 `snapshot` 值为 `true` 。

设置截图的大小和质量

找开发要单位通过 `setSanpshotProperties(width,height,rate)` 方法设置截取图片的大小和图片质量，大小默认为100%。示例如下：

```
//将截图的宽度、高度、和质量分别设置为300、200、0.9
//高度、宽度单位为px，质量可取值0-1之间的数字，默认是1
player.setSanpshotProperties(300,200,0.9)
```

订阅截图事件

截图完成时会触发 `snapshoted` 事件，并返回截图数据。示例如下：

```
player.on("snapshoted", function(data) {
     console.log(data.paramData.time);
     console.log(data.paramData.base64);
     console.log(data.paramData.binary);
 });
```

参数说明：

- time：截图的视频播放时间点。
- base64：所截图的base64串。可以直接用于 `img` 显示。
- binary：所截图的二进制数据。可以用于上传。

截图水印

H5模式可设置 `snapshotWatermark` 属性为截图添加水印。属性参数如下：

参数	说明
left	到左边的距离。
top	左上角的高度，会包含文字的高度。
text	水印文字。
font	设置文字格式：可以多个属性一起设置，中间空格隔开。 • font-style：设置字体风格。 • font-weight：字体粗细设置。 • font-size：设置字体大小。 • font-family：设置字体。
strokeColor	设置用于笔触的颜色。
fillColor	填充绘画的颜色。

示例如下：

```
snapshotWatermark:{
    left:"100",
    top:"100",
    text:"测试水印",
    font:"italic bold 48px 宋体",
    strokeColor:"red",
    fillColor:'green'
  }
```

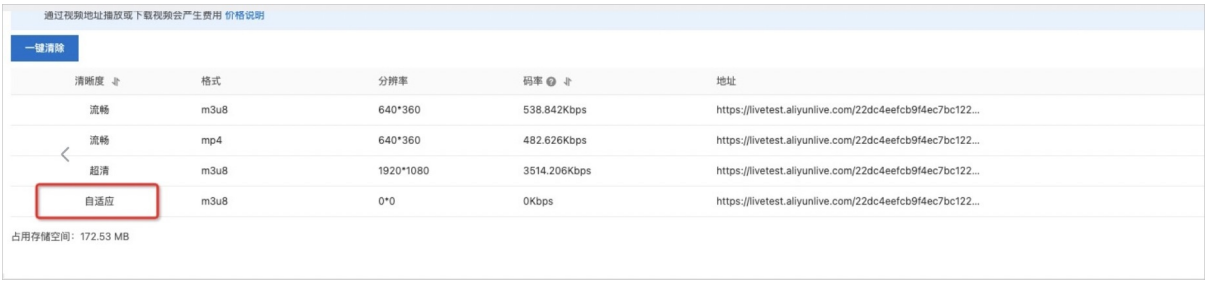
长视频场景

多码率HLS网络自适应切换 开发输入

多码率指的是source只传入一个多码率的地址（a master playlist），然后指定参数 `isVBR:true` 。多码率支持根据网络环境自适应切换视频清晰度，也支持手动切换。

获取当前播放地址

多码率：player._hls.levels[player._hls.currentLevel]

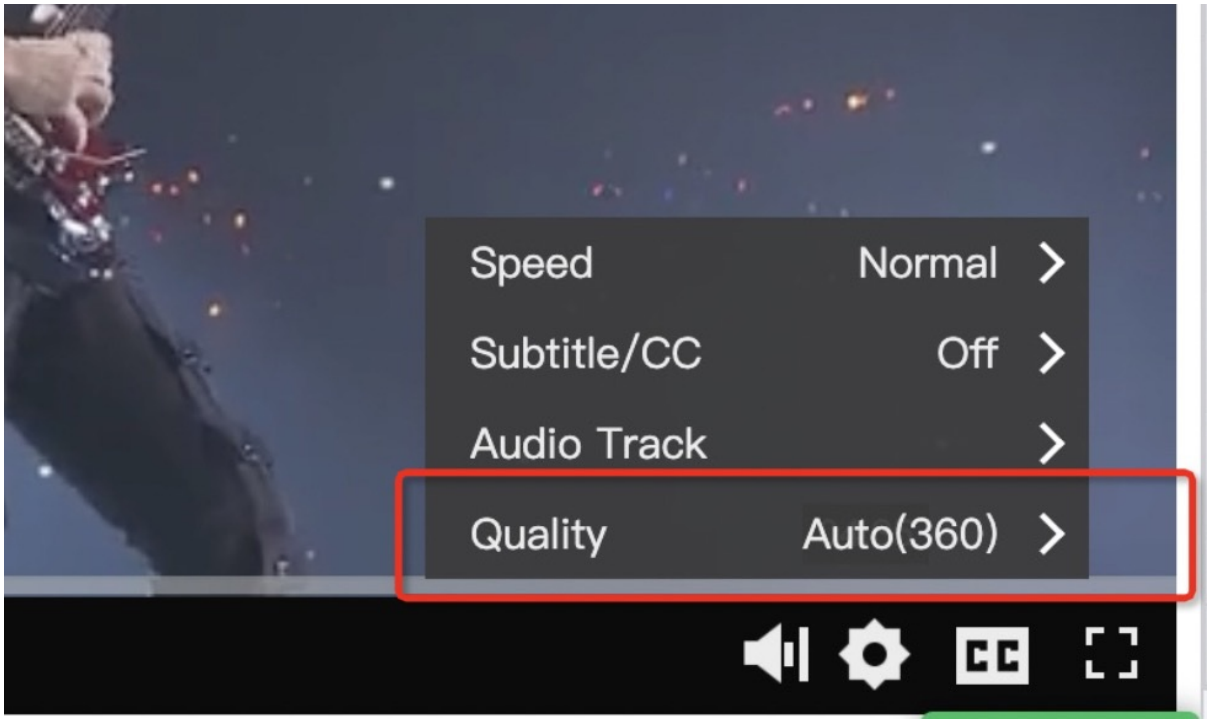


> **说明**
>
> - Safari浏览器自动（Auto）模式下不支持显示当前码率。
> - HLS视频流需要经过多码率视频转码模板组打包处理，可以在点播控制台的 **配置管理** > **媒体处理配置** > **转码模板组** 中配置生产对应的视频流。具体操作，请参见 **视频或字幕打包模板设置** 。

示例代码如下：

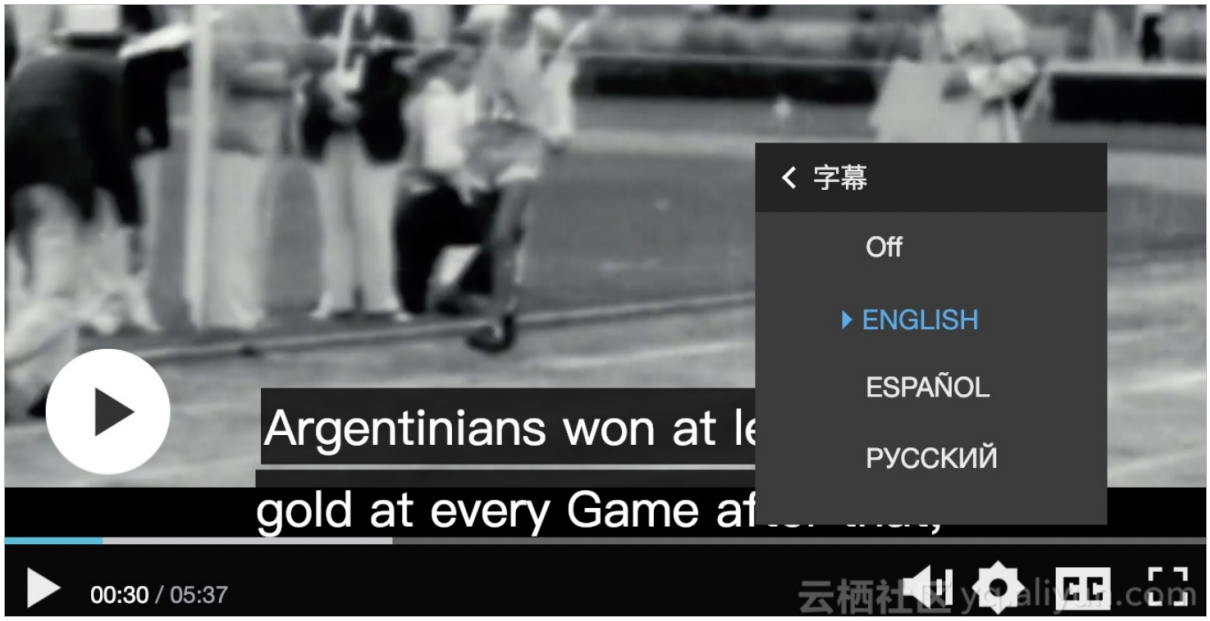
```
varplayer = newAliplayer({
 "id":"player-con",
 "source":"自适应url地址",
 "isVBR":true,
 },
 function () { }
);
```

配置完成的效果图如下：



外挂字幕开发输入

控制台配置，Web播放器SDK无需额外设置（只支持Vid+PlayAuth播放方式播放）。

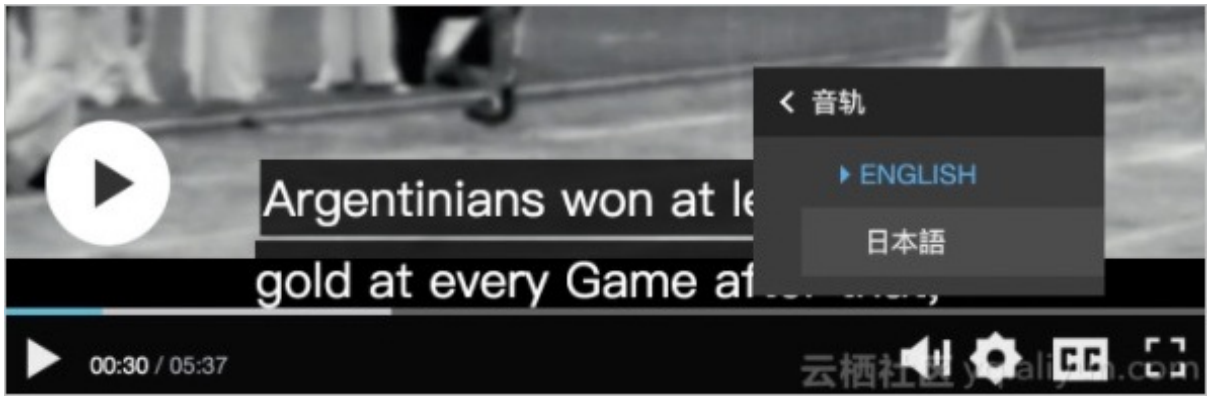


Web播放器SDK除了提供默认UI操作外， 还提供了CCService满足用户的一些自定义需求，比如需要根据浏览器的语言默认播放那个语言等等，通过 `player._ccService` 属性访问字幕服务，字幕服务提供了如下的API：

函数名称	参数	说明
switch	language	切换字幕
open	不涉及	开启字幕open=关闭？要不要找找开发确认一下
close	不涉及	关闭字幕
getCurentSubtitle	不涉及	获取当前字幕的language的值

多音轨开发输入

播放器无需额外设置。效果图如下：



多语言开发输入

Web播放器SDK默认支持中文和英文，并且依赖于浏览器的语言设置自动启用中文或英文资源。除了支持这两种资源外，还提供自定义语言的能力，支持其他国际语言。另外Web播放器SDK还支持点播服务的多地域，可以支持Vid+playauth播放方式播放东南亚和欧洲的视频资源。

语言设置属性

Web播放器SDK提供language属性用于指定语言，此属性的优先级高于浏览器的语言设置，默认值为空，示例代码如下：

```
var player = new Aliplayer({
    id: "player-con",
    source: "",
    width: "100%",
    height: "500px",
    autoplay: true,
    language: "en-us",
  }, function (player) {
    console.log("播放器创建成功");
  });
```

英文版本播放器

```
var player = new Aliplayer({
 "id": "player-con",
 "source": "",
 "language": "en-us" //zh-cn：中文。en-us：英文。
 },
 function (player) {}
);
```

自定义语言

当需要支持中文和英文之外的语言时，需要使用自定义语言的功能， 这时可以通过 `languageTexts` 属性指定语言资源属性， `languageTexts` 使用对象字面量的形式， `language` 属性的值为key，JSON value值为指定语言翻译的资源内容。示例代码如下：

> **说明** 如果不能确定哪些资源需要翻译，可以借助在线的翻译资源输入工具确认。单击进入 **在线配置工具**，在顶部导航栏选择 **更多配置** > **语言**，选择或输入语言Key后，会弹出一个语言翻译页面。在这个页面上可以将需要翻译的资源翻译为对应的语言，提交后会生成代码。

```
var player = new Aliplayer({
 "id": "player-con",
 "source": "",
 "language": "CustomLanguage",//自己命名，string类型即可
"languageTexts":{
    "CustomLanguage":{
        "Pasue":"CustomLanguage暂停"
        //其他播放器的字段，参考
https://player.alicdn.com/lang.json? spm=a2c4g.11186623.0.0.5a746515vnwUSi&file=lang.json

            }
        }
    },
    function (player) {}
);
```

> **说明**

支持多地域播放
当前点播服务支持下面的地域：上海、法兰克福、新加坡，Web播放器SDK的Vid+playauth播放和STS播放支持多地域的播放，当知道用户需要播放哪个地域的视频时，播放器会调用此地域的点播服务，获取视频的播放地址。

- Vid+playauth播放：从playauth里面解析出Region，获取对应地域的视频，因此不用特意再指定播放那个地域的视频。
- STS播放：通过播放器提供的Region属性指定播放哪个地域的视频，Region默认值为 `'cn-shanghai'` ，可选值包含：cn-shanghai、eu-central-1、ap-southeast-1，示例代码如下：

```
var player = new Aliplayer({
    id: "player-con",
    width: "100%",
    height: "500px",
    autoplay: true,
    language: "en-us",
    vid : '1e067a2831b641db90d570b6480f****',
    accessKeyId: '',
    securityToken: '',
    accessKeySecret: ''
    region:'eu-central-1',//法兰克福地域
  }, function (player) {
    console.log("播放器创建成功");
  });
```

H5模式直播时移

2021.12.8：H5直播时移的内容因功能不完善，问题较多，用户体验较差，暂时屏蔽掉。待后续功能完善后再优化对外。相关Aone：web播放器，直播时移内容暂时下调 (alibaba-inc.com) 。

- 开通直播时移
  - 直播时移功能首先需要到阿里云直播服务里开通，更多信息，请参见 直播时移。
  - 播放器开启时移功能时需要设置如下属性：

名称	说明
isLive	设置值为：true。
liveTimeShiftUrl	时移信息查询URL。
liveStartTime	直播开始时间。
liveOverTime	直播结束时间。
liveShiftSource	直播时移hls地址。 说明 只有在source为flv直播流是需要设置。
liveShiftMinOffset	时移切片录制需要时间，seek时间过于接近直播时间会因为切片未生成而 404，默认会有一个最小seek时间，可以通过此参数配置，单位是秒，默认值30（10秒一个切片，保证至少有3个切片）。

- 直播时移UI
  直播时移的UI构成主要是可时移区域的进度条，时间显示：

> **说明** 时间区域从左到右分别为：当前播放时间、直播结束时间、当前直播时间。

- **调整直播结束时间**
  在播放过程中可以通过调用 `liveShiftSerivce.setLiveTimeRange` **方法调整直播开始/结束时间时，UI会做相应的变化。示例如下：**

```
player.liveShiftSerivce.setLiveTimeRange("",'2018/01/04 20:00:00')
```

- **FLV直播和HLS时移切换**
  由于延时的原因，请在直播流用flv**格式，时移流用**hls**格式。**
  Web**播放器SDK支持模式：**
  - source**属性指定**flv**直播地址。**
  - liveShiftSource**属性指定**hls**的地址。**

  示例如下：

```
{
 source:'http://localhost/live****/example.flv',
 liveShiftSource:'http://localhost/live****/example.m3u8',
}
```

4.6. 插件组件

本文介绍H5**模式的**Web**播放器SDK的常用组件，以及自定义组件的实现。**

内置组件

> **说明** 组件功能演示请参见 功能演示。

记忆播放
自动记忆用户上一次播放的视频位置，记忆播放组件分两种，一种点击播放（会提示用户上次播放的位置，用户可以点击seek**），另一种是自动播放（自动**seek**到上次播放位置）。更多信息请参见记忆播放组件。**

跑马灯
主要用于一些滚动文字，可以自定义出现的位置，文字颜色和字体大小。更多信息，请参见 跑马灯组件。

开始广告（图片）
视频即将开始播放时显示的图片广告。更多信息，请参见 开始广告组件。

暂停广告（图片）
暂停视频时在播放器的中间显示图片广告。更多信息，请参见 暂停广告组件。

多视频广告
在视频即将播放时，实现多视频广告。更多信息，请参见 多视频广告组件。

播放下一个
在Web**播放器SDK的控制条上添加** `播放下一个` **视频按钮，按钮的点击事件用户可以自定义并作为组件参数传入。更多信息，请参见播放下一个组件。**

播放列表
在Web**播放器SDK控制条上增加视频列表，视频切换的按钮，并在播放器上显示视频列表。更多信息，请参见播放列表组件。**

旋转和镜像
在Web**播放器SDK控制条上添加旋转和镜像按钮，用于视频旋转和镜像变换。更多信息，请参见 旋转和镜像组件。**

视频广告

> **说明** 移动端浏览器可能劫持Web播放器SDK播放视频的行为，导致部分功能不可用。

在视频即将播放时，限时的视频广告。更多信息，请参见 视频广告组件 。

弹幕

> **说明** 本组件集成了CommentCoreLibrary弹幕库，更多信息请参见 CommentCoreLibrary文档 。

在控制条上增加了弹幕关闭和开启按钮，以及发送弹幕的输入框，和发送弹幕的按钮。更多信息，请参见 弹幕组件 。

试看
用于用户试看，试看结束后提示用户，观看完整版的条件。更多信息，请参见 试看组件 。

进度条标记
对视频的关键点进行打点，鼠标移动到关键点时能够查看当前用户设置的图片等标注信息。更多信息，请参见进度打点组件 。

字幕
用于快速切换不同语言的字幕。更多信息，请参见 字幕组件 。

音轨
用于切换不同版本的音轨。更多信息，请参见 音轨组件 。

引入组件

1. 引入Web播放器SDK组件。

```
<scripttype="text/javascript"charset="utf-8"src="/aliplayercomponents-
1.0.8.min.js"></script>
```

> **说明** 目前没有CDN资源，只能本地引入。下载请参见 播放器组件下载 。

2. Web播放器SDK挂载组件。

> **说明** 本文仅提供示例代码，不同组件细节不同，需要具体配置。

```
var player = new Aliplayer({
        id: "player-con",
        source: "//player.alicdn.com/video/editor.mp4",
        components: [{
            name: 'xxxComponent',
            type: AliPlayerComponent.xxxComponent
        }]
    }, function (player) {
 });
```

组件使用

当只需要单独的某个组件时，只需要引用对应组件的JS文件，如果引用的是Web播放器SDK组件库文件，则通过AliPlayerComponent.XXX引用具体的组件。

打包组件

如果是Windows环境， `NODE_ENV` 环境变量的设置和在macOS和Linux下的设置不同，要将 *package.json* 下的 `build_customize` 命令更改为：

```
"build_customize": set NODE_ENV=production&&webpack --config
webpack.config.customize.js --progress
```

- **打包全部组件**
  下面的两个指令中的任意一个都可以打包全部Web播放器SDK组件，打包之后的文件是 `/disk/aliplayer-components/aliplayercomponents.min.js` **。**

```
$ npm run build
# 或者
$ npm run build all
```

- **打包自定义组件**
  为了减少体积，用户可以自己选择需要打包的组件，只要执行以下命令：

```
$ npm run build componentsName # componentsName 是组件名称
```

> **说明** componentsName 是组件的名称，多个组件名称以空格隔开，例如 `$ npm run build AliplayerDanmu BulletScreen # 打包弹幕组件和跑马灯组件` ，componentsName 可选的值有：
>
> - AliplayerDanmu：弹幕组件
> - BulletScreen：跑马灯组件
> - MemoryPlay：记忆播放组件
> - PauseAD：暂停广告（图片）组件
> - PlayerNext：播放下一个视频组件
> - Playlist：播放列表组件
> - Preview：试看组件
> - RotateMirror：旋转镜像组件
> - StartAD：开始广告（图片）组件
> - Staticad：静态广告（图片）组件
> - VideoAD：视频广告组件
> - Caption：字幕组件
> - Track：音轨组件
> - ManyVideoAD：多视频广告组件

打包之后的文件是 `/disk/aliplayer-components/aliplayercomponents.min.js` **，引用到用户的页面中即可。**

引用具体的组件

- **在HTML文件中引用具体的JS文件**

```
<script type="text/javascript" src="js/staticAdComponent.min.js"></script>
```

- **给Web播放器SDK注入组件**

```
var option = {
     id: "J_prismPlayer",
     autoplay: true,
     isLive:false,
     playsinline:true,
     width:"100%",
     height:"100%",
     useH5Prism:true, //启用H5播放器
     useFlashPrism:false,
     source:source,
     vid:vid,
     playauth:playauth,
     cover:"",
     components:[{type:StaticAdComponent,args:
['http://cdnoss.example.com/cover****.png',
     'http://player.alicdn.com']}]
};
var player = new Aliplayer(option);
```

引入Web播放器SDK组件库

- **在HTML文件中引用具体的JS文件**

```
<script type="text/javascript" src="js/aliplayerComponents.min.js"></script>
```

- **给Web播放器SDK注入组件**

名称	说明
name	组件名称，可以通过名称得到组件
type	组件类型
args	组件的参数

```
var option = {
      id: "J_prismPlayer",
      autoplay: true,
      isLive:false,
      playsinline:true,
      width:"100%",
      height:"100%",
      useH5Prism:true, //启用H5播放模式
      useFlashPrism:false,
      source:source,
      vid:vid,
      playauth:playauth,
      cover:"",
      components:[{type:AliPlayerComponent.StaticAdComponent,args:
['http://cdnoss.youkouyang.com/cover.png',
      'http://player.alicdn.com']},
      notParameComponent,
      {name:'adComponent1',type:notParameComponent}
      ]
};
var player = new Aliplayer(option);
```

获取组件
提供getComponent方法获取实例组件，参数为组件的名字。

```
var component = player.getComponent('adComponent');
```

安装依赖项
本Demo使用了ES6、webpack、gulp等技术。

- Node.js
- Webpack4.0
- gulp

```
$ cd customComponents
$ npm install
```

4.7. 接口说明

本文对Web播放器SDK的属性、方法、事件和错误码进行了说明，并提供了播放器接口的示例代码。

属性

2021.12.10：用户反馈问题在常见问题中都有说明，此处添加常见问题链接

> **说明** 如果您在使用过程中遇见问题，可以参考 常见问题或提交工单联系阿里云客服解决。

2021.12.8：直播时移的内容因功能不完善，问题较多，用户体验较差，暂时屏蔽掉时移相关接口。待后续功能完善后再优化对外。相关Aone：web播放器，直播时移内容暂时下调 (alibaba-inc.com) 。

名称	类型	说明
id	String	播放器外层容器的dom元素ID。

名称	类型	说明
source	String	视频播放地址URL，单独URL。默认状态，表示使用 `vid+playauth` 。 **说明** URL播放方式优先级最高。URL播放方式支持多清晰度设置：source:'{"HD":"address1","SD":"address2"}'，更多信息，请参见多清晰度播放。
vid	String	媒体转码服务的媒体ID。
playauth	String	播放权证，获取播放权重请参见 获取音视频播放凭证。
height	String	播放器高度，取值： • 100% • 100px **说明** Chrome浏览器下Flash播放器分别不能小于397x297px。
width	String	播放器宽度，取值： • 100% • 100px **说明** Chrome浏览器下Flash播放器分别不能小于397x297px。
videoWidth	String	视频宽度，仅H5模式支持。更多信息，请参见 设置显示模式。
videoHeight	String	视频高度，仅H5模式支持。更多信息，请参见 设置显示模式。
preload	Boolean	播放器自动加载，目前仅H5模式可用。
cover	String	播放器默认封面图片，请填写正确的图片URL地址。需要autoplay值为false时，才生效。Flash播放器封面也需要开启允许跨域访问。
isLive	Boolean	播放内容是否为直播，直播时会禁止用户拖动进度条。
autoplay	Boolean	播放器是否自动播放，在移动端autoplay属性会失效。 **说明** Safari11不支持自动播放，如果需要，可通过右键单击浏览器地址栏，选择此网站的设置 > 允许全部自动播放 来设置。
rePlay	Boolean	播放器自动循环播放。

名称	类型	说明
useH5Prism	Boolean	指定使用H5播放器。
useFlashPrism	Boolean	指定使用Flash播放器。
playsinline	Boolean	H5是否内置播放，有的Android浏览器不起作用。
showBuffer	Boolean	显示播放时缓冲图标，默认值为 true。
skinRes	Url	皮肤图片，不建议随意修改该字段，如要修改，更多信息，请参见设置播放器皮肤 。
skinLayout	Array | Boolean	功能组件布局配置，不传该字段使用默认布局。取值：false隐藏所有功能组件。更多信息，请参见 配置skinLayout属性。
controlBarVisibility	String	控制面板的实现，默认值为： hover。取值： • click：点击。 • hover：停留。 • always：一直。 • never：隐藏整个控制面板。
showBarTime	String	控制栏自动隐藏时间，单位毫秒。
extraInfo	String	JSON串用于定制性接口参数，目前仅Flash支持，取值： • fullTitle：测试页面，全屏时显示视频标题。 • m3u8BufferLength：播放HLS文件时加载缓存，ts文件长度单位为秒。
enableSystemMenu	Boolean	是否允许系统右键菜单显示，默认为 false。
format	String	指定播放地址格式，只有使用vid的播放方式时支持可选值，取值： • mp4 • hls • flv • mp3 默认为空，仅H5支持。
mediaType	String	指定返回音频还是视频，只有使用vid的播放方式时支持，默认值为video。取值： • video：视频。 • audio：针对只包含音频的视频格式，比如音频的MP4。仅H5支持。
qualitySort	String	指定排序方式，只有使用Vid + PlayAuth播放方式时支持。取值： • desc：表示按倒序排序（即：从大到小排序）。 • asc：示按正序排序（即：从小到大排序）。 默认值：asc，仅H5支持。

名称	类型	说明
definition	String	显示视频清晰度，多个使用半角逗号（,）分隔，比如：‘FD,LD’，此值是vid对应流清晰度的一个子集，仅H5模式支持。取值： • FD（流畅） • LD（标清） • SD（高清） • HD（超清） • OD（原画） • 2K（2K） • 4K（4K）
defaultDefinition	String	默认视频清晰度，此值是vid对应流的一个清晰度，仅H5模式支持。取值： • FD（流畅） • LD（标清） • SD（高清） • HD（超清） • OD（原画） • 2K（2K） • 4K（4K）
x5_type	String	声明启用同层H5播放器，启用时取值： H5。更多信息，请参见如何启用H5的同层播放 。
x5_fullscreen	Boolean	声明视频播放时是否进入到TBS的全屏模式，取值： • false：不把视频做为背景。 • true：把视频做为背景。 默认值为false。 更多信息，请参见 如何启用H5的同层播放 。
x5_video_position	String	声明视频播在界面上的位置，默认值为 center。取值： • center：居中。 • top：顶部。 更多信息，请参见 如何启用H5的同层播放 。
x5_orientation	String	声明TBS播放器支持的方向，取值： • landscape：横屏。 • portraint：竖屏。 更多信息，请参见 如何启用H5的同层播放 。
x5LandscapeAsFullScreen	String	声明TBS全屏播放是否横屏，默认值为 true。取值： • true：横屏。 • false：竖屏。
autoPlayDelay	Number	延迟播放时间，单位：秒。更多信息，请参见 配置延迟播放。

名称	类型	说明
autoPlayDelayDisplayText	String	延迟播放提示文本，更多信息，请参见 配置延迟播放。
language	String	国际化，默认为zh-cn。如果未设置，则采用浏览器语言。取值： • zh-cn：中文。 • en-us：英文。
languageTexts	JSON	自定义国际化文本JSON结构，key的值需要和language属性值对应起来。示例：{jp:{Play:"Play"}} 自定义值请参见JSON结构。
snapshot	Boolean	是否启用Flash截图功能。取值： • true：启用。 • false：禁用。（默认值）
snapshotWatermark	Object	H5设置截图水印。
useHlsPluginForSafari	Boolean	Safari浏览器是否启用HLS插件播放，Safari 11除外。取值： • true：启用。 • false：禁用。（默认值）
enableStashBufferForFlv	Boolean	H5播放FLV时，设置是否启用播放缓存，只在直播下起作用。取值： • true：启用。（默认值） • false：禁用。
stashInitialSizeForFlv	Number	H5播放FLV时，初始缓存大小，只在直播下起作用。默认32KB。 当设置的值较小时，会提升起播速度，但是值太小时，可能会导致播放一小段之后卡顿。
loadDataTimeout	Number	缓冲多长时间后，提示用户切换低清晰度，单位：秒。默认20秒。
waitingTimeout	Number	最大缓冲超时时间，超过这个时间会有错误提示，单位：秒。默认60秒。
diagnosisButtonVisible	Boolean	是否显示检测按钮，取值： • true：显示按钮。 • false：不显示按钮。 默认值为true。
disableSeek	Boolean	禁用进度条的Seek，取值： • true：禁用。 • false：不禁用。 默认值为false。 说明 仅Flash支持。

名称	类型	说明
encryptType	int	加密类型，播放点播私有加密视频时，默认值为 0，取值： • 0：不加密视频。 • 1：播放加密视频。 说明 • 私有加密采用VID+Playauth的方式进行播放。 • Web端标准加密使用URL方式播放，请参见视频加密播放。
progressMarkers	Array	进度条打点内容数组，更多信息，请参见 进度条标记。
vodRetry	int	点播失败重试次数，默认3次。
liveRetry	int	直播播放失败重试次数，默认5次。
hlsFrameChasing	boolean	HLS直播模式下，是否开启追帧。取值： • true：开启追帧。 • false：不开启追帧。（默认值）
chasingFirstParagraph	number	第一段追帧，单位：秒。默认20秒。
chasingSecondParagraph	number	第二段追帧，单位：秒。默认40秒。
chasingFirstSpeed	number	第一段追帧的倍速，默认1.1倍速。
chasingSecondSpeed	number	第二段追帧的倍速，默认1.2倍速。
flvFrameChasing	Boolean	FLV直播模式下，是否开启追帧，取值： • true：开启追帧。 • false：不开启追帧。 默认值为false。
keyShortCuts	Boolean	是否启用快捷键，取值： • true：开启快捷键。 • false：不开启快捷键。 默认值为false。 说明 方向键（左右键）控制快进和快退，方向键（上下键）控制音量的增减，空格键暂停和播放。
keyFastForwardStep	Number	快进快退的时间长度，单位：秒。默认10秒。

方法

方法需要在ready事件发生之后或创建播放器ready回调里，H5模式下可以在创建播放器构造函数的回调函数里调用。示例如下：

- H5播放器

```
//H5 播放器
 var player = new Aliplayer({},function(player) {
    player.play();
 });
```

- Flash播放器

```
//Flash 播放器
 player.on('ready',function(e) {
    player.play();
 });
```

名称	参数	说明
play	无	播放视频。
pause	无	暂停视频。
replay	无	重播视频。
seek	time	跳转到某个已加载的时刻进行播放，时间单位：秒。
getCurrentTime	无	获取当前的播放时刻，返回的时间单位：秒。
getDuration	无	获取视频总时长，返回的单位为秒，这个需要在视频加载完成以后才可以获取到，可以在play事件后获取。
getVolume	无	获取当前的音量，返回值为 0~1的实数。iOS和部分Android会失效。
setVolume	无	设置音量，vol为0~1的实数，iOS和部分Android会失效。
loadByUrl	url，time	直接播放视频url，time为可选值（单位：秒）。目前只支持同种格式（MP4、FLV、HLS）之间切换。暂不支持直播RTMP流切换。
replayByVidAndPlayAuth	vid：视频ID，playauth：播放凭证	目前只支持H5播放器。暂不支持不同格式视频间的之间切换。暂不支持直播RTMP流切换。 可用于点播DRM流的切换，用法：`player.replayByVidAndPlayAuth(vid)`。
replayByVidAndAuthInfo	仅MPS用户时使用参数顺序为：vid、accId、accSecret、stsToken、authInfo、domainRegion	目前只支持H5播放器。暂不支持不同格式视频间的之间切换。暂不支持直播rtmp流切换。
setPlayerSize	w，h	设置播放器大小，取值： • 400px • 60% Chrome浏览器下Flash播放器分别不能小于397x297px。

名称	参数	说明
setSpeed	speed	手动设置播放的倍速，支持0.5~2倍速播放，倍速播放仅H5模式支持。移动端可能会失效，比如Android微信。倍速播放UI默认是开启的。2022.03.10：新增倍速范围 说明 关掉倍速的方法： • 目前无法单独关闭或者自定义倍速，只能整体关掉设置。 • 通过hack方式关掉倍速是样式覆盖： `.prism-setting-speed { display: none !important; }`
setSanpshotProperties	width：宽度，height：高度，rate：截图质量	设置截图参数。
fullscreenService.requestFullScreen	无	播放器全屏，仅H5支持。
fullscreenService.cancelFullScreen	无	播放器退出全屏，iOS调用无效，仅H5支持。
fullscreenService.getIsFullScreen	无	获取播放器全屏状态，仅H5支持。
getStatus	无	获取播放器状态，取值： • init：初始化。 • ready：准备。 • loading：加载中。 • play：播放。 • pause：暂停。 • playing：正在播放。 • waiting：等待缓冲。 • error：错误。 • ended：结束。
setRotate	rotate：旋转角度	参数为旋转角度，正数为正时针旋转，负数为逆时针旋转。示例：setRotate(90)。更多信息，请参见设置显示模式。
getRotate	无	获取旋转角度。更多信息，请参见 设置显示模式 。
setImage	image：镜像类型	设置镜像，取值： • horizon：水平。 • vertical：垂直。 示例：setImage('horizon')。更多信息，请参见 设置显示模式。
dispose	无	播放器销毁。

名称	参数	说明
setCover	cover封面地址	设置封面。
setProgressMarkers	markers打点数据集合	设置打点数据。
setPreviewTime	time试看时间	设置试看时间，单位：秒。更多信息，请参见 试看。
getPreviewTime	无	获取试看时间。
isPreview	无	是否试看。
getCurrentPDT	无	HLS的视频格式支持实时获取ProgramDateTime。

事件

播放器事件

名称	说明
ready	播放器视频初始化按钮渲染完毕。播放器UI初始设置需要此事件后触发，避免UI被初始化所覆盖。 说明 播放器提供的方法需要在该事件发生后才可以调用。
play	视频由暂停恢复为播放时触发。
pause	视频暂停时触发。
canplay	能够开始播放音频和视频时发生，会多次触发，仅H5播放器。
playing	播放中，会触发多次。
ended	当前视频播放完毕时触发。
liveStreamStop	直播流中断时触发。HLS、FLV、RTMP在重试5次未成功后触发。提示上层流中断或需要重新加载视频。 说明 如果HLS、FLV、RTMP的直播流断流或者出错，播放器会自动重试5次，不需要上层添加重试逻辑。
onM3u8Retry	HLS直播流中断后重试事件，每次断流只触发一次。
hideBar	控制栏自动隐藏事件。
showBar	控制栏自动显示事件。
waiting	数据缓冲事件。
timeupdate	播放位置发生改变时触发，仅H5模式播放器。可通过getCurrentTime方法，得到当前播放时间。
snapshoted	截图完成事件。
requestFullScreen	全屏事件，仅H5模式支持。

名称	说明
cancelFullScreen	取消全屏事件，iOS下不会触发，仅H5模式支持。
error	错误事件。
startSeek	开始拖拽，参数返回拖拽点的时间。
completeSeek	完成拖拽，参数返回拖拽点的时间。
resolutionChange	直播情况下，推流端切换了分辨率。
seiFrame	HLS或FLV收到SEI消息。

订阅事件

- 通过播放器实例的on方法订阅。示例如下：

```
var handleReady = function(e)
{
    console.log(e);
}
player.on('ready',handleReady);
```

- 通过播放器实例的off方法取消订阅。示例如下：

```
player.off('ready',handleReady);
```

错误码

代码	含义
4001	参数不合理。
4002	鉴权过期。
4003	无效地址。
4004	地址不存在。
4005	开始下载数据错误，检测网络情况或播放地址是否可以访问。
4006	开始下载元数据错误。
4007	播放时错误。
4008	加载超时，检测网络情况或播放地址是否可以访问。
4009	请求数据错误，测网络情况或播放地址是否可以访问。
4010	不支持阿里云视频加密（私有加密）播放。
4011	播放格式不支持。
4012	playauth解析错误。
4013	播放数据解码错误MEDIA_ERR_DECODE，检测浏览器是否支持视频格式。

代码	含义
4014	网络不可用。
4015	获取数据过程被中止MEDIA_ERR_ABORTED。
4016	播网络错误加载数据失败MEDIA_ERR_NETWORK。
4017	返回的播放地址为空。
4100	信令请求失败。
4110	不支持webrtc。
4111	不支持此浏览器。
4112	浏览器版本过低。
4113	不支持H264。
4114	create offer失败。
4115	自动播放失败。
4116	播放URL协议错误。
4118	订阅时候传入的HtmlElement既不是Audio也不是Video。
4200	播放失败。
4400	未知错误MEDIA_ERR_SRC_NOT_SUPPORTED（由于服务器或网络原因不能加载资源，或者格式不支持）。
4500	服务端请求错误，查看Network里点播服务的请求的具体错误。

4.8. 常见问题

本文针对Web播放器使用过程中常见的问题提出解决方案或规避措施。

如何正确选择播放模式

播放器包含H5、Flash、自适应播放器，建议用户选择自适应播放器，可以根据终端类型、浏览器类型和地址协议选择最合适的播放器。具体操作，请参见在线配置。

自适应播放器

根据终端类型、浏览器类型、设置的属性和地址协议选择最合适的播放器。

- useFlashPrism=true、rtmp和http-flv协议时，采用Flash播放。
- 移动端采用H5播放。
- useH5Prism=true，采用H5播放。
- PC端MP4采用H5播放。
- PC端如果浏览器或通过Web播放器的插件播放HLS视频，则采用H5播放，否则采用Flash播放。
- 其它都用H5播放。

> **说明** 适配的基本原则是：H5优先级最高，能H5播放的绝不选择Flash，除非用户指定用Flash播放。

H5播放器

- **如何手工启用H5播放器**
  手工启用H5播放器有两种方式：
  - 直接引用H5播放器的js文件。
  - 使用自适应播放器，然后设置useh5Prism值为 **true**。

- **H5播放器如何初始播放位置**
  H5播放器，示例如下：

```
var seeked = false;
player.on('canplaythrough',function  (e) {
  if(!seeked)
  {
    seeked = true;
    player.seek(100);
  }
});
```

- **H5播放器如何切换vid和playauth**
  H5播放器，直接调用 `replayByVidAndPlayAuth` 方法。示例如下：

```
 player.replayByVidAndPlayAuth(newVid, newPlayAuth)
```

- **如何调整H5播放器的播放按钮的大小和位置**
  - 重写CSS，比如减小一倍。示例如下：

```
 .prism-player .prism-big-play-btn {
    width: 45px;
    height: 45px;
    background-size: 128px 256px;
}
```

  - 位置可以通过设置skinLayout里bigPlayButton的x,y属性。示例如下：

```
skinLayout: [
    {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {
      name: "H5Loading", align: "cc"
    },
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
        {name: "progress", align: "tlabs", x: 0, y: 0},
        {name: "playButton", align: "tl", x: 15, y: 26},
        {name: "timeDisplay", align: "tl", x: 10, y: 24},
        {name: "fullScreenButton", align: "tr", x: 20, y: 25},
        {name: "volume", align: "tr", x: 20, y: 25},
      ]
    }
  ]
```

- **调用seek方法后，播放器无法实现暂停按钮** 2022.03.08新增
  按钮状态取决于播放器原来的状态，想要实现暂停按钮，可以在seek后再调用 `player.pause()` 方法。

Flash播放器

- **如何手工启用Flash播放器**
  播放器基本都支持Flash播放，最新的一些浏览器会禁用Flash，需要手工启用，参考下面链接：
  - IE使用说明
  - Firefox使用说明
  - Chrome使用说明

  如需要手工启用Flash播放器，有以下两种方式：
  - 直接引用Flash播放器的js文件。
  - 使用自适应播放器，然后设置useFlashPrism值为 **true**。

- **Flash播放器对mp4/flv无法拖拽**
  mp4与flv拖拽需要CDN添加支持，是通过播放器发送带时间的请求到CDN，CDN返回该时间段的视频数据。如果要实现拖拽，需要以下两个条件：
  - 文件索引信息需要在视频的头部，mp4包含视频时间戳等索引信息，以及flv的meta信息要在视频最前面，播放器解析到视频索引信息后，才可以依据拖拽的位置通过索引信息拿到指定位置的数据点，去向CDN发送请求。
  - CDN支持带时间/byte range的请求，需要在CDN控制台开启，请参见 配置拖拽播放 。

- **Flash播放器如何初始播放位置**
  Flash播放器，示例如下：

```
var seeked = false;
player.on('loadedmetadata',function  (e) {
  if(!seeked)
  {
    seeked = true;
    player.seek(20);
  }
});
```

- **Flash播放器播放m3u8提示跨域错误**
  播放器跨域访问时需要添加策略文件，即在视频播放链接所在域名的根目录下，添加crossdomain.xml文件，其中添加播放器所在域名的权限。示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<cross-domain-policy>
    <allow-access-from domain="*"/>
    <allow-http-request-headers-from domain="*" headers="*" secure="false"/>
</cross-domain-policy>
```

- **Flash播放器封面图片无法显示**
  - 确认cover字段输入URL是否有效。
  - 确认cover输入的URL所在域名是否存在有效的crossdomain.xml文件。

- **Flash播放器如何切换vid和playauth**
  Flash播放器中需要销毁，根据新的vid和playauth重新创建一个。示例如下：

```
//销毁
    FlashPlayer.dispose();
    $('#FlashPlayer').empty();
    //重新创建
    FlashPlayer = new Aliplayer({
           id: 'FlashPlayer',
           autoplay: true,
           playsinline:true,
           vid: newVid,
           playauth: newPlayAuth,
           useFlashPrism:true
       });
```

其它通用问题

- **解决Android微信上自动弹出全屏播放**
  Android系统手机在微信和QQ浏览器里自动全屏播放，这是腾讯浏览器的内置行为不能修改，原因是由于腾讯浏览器挟持了video标签，由腾讯内置的播放器播放视频，但可以启用同层播放功能，可以解决视频覆盖Dom元素的问题，请参见同层播放。

> **说明** 全屏播放后可通过input设置绝对定位来唤起软键盘 。 用户反馈，添加说明。无需送翻。

- **在iOS微信里如何自动播放**
  iOS端微信自动播放。示例如下：

```
<script src="http://res.wx.qq.com/open/js/jweixin-1.0.0.js"></script>
<script>
function autoPlay() {
            wx.config({
                // 配置信息，即使不正确也能使用 wx.ready
                debug: false,     //false代表关闭调试模式，ture代表开启调试模式
                appId: '',        //appId
                timestamp: 1,     //生成签名的时间戳
                nonceStr: '',     //生成签名的随机串
                signature: '',    //签名
                jsApiList: []     //需要使用的JS接口列表
            });
            wx.ready(function() {
                if(player)//创建的Aliplayer对象
                {
                  player.play();
                }
            });
    };
    // 解决ios不自动播放的问题
    autoPlay();
</script>
```

- **如何定时获取播放时间**
  通过定时器每秒调用播放器的getCurrentTime方法获取播放时间，在暂停、出错和结束播放时清除定时器。示例如下：

```
 var timer = null;
function getTime()
{
   var currentTime = player.getCurrentTime();
   //to do
   timer = setTimeout(getTime,1000);
}
//清除定时器
function clear()
{
   if(timer)
   {
      clearTimeout(timer);
      timer = null;
   }
}
player.on('ended',function  (e) {
  clear();
 });
player.on('pause',function  (e) {
  clear();
 });
player.on('error',function  (e) {
  clear();
 });
```

- **手机端禁止自动全屏播放**
  - **手机端不希望全屏播放，iOS可以设置属性playsinline的值为： true。**
  - **Android系统手机在微信和QQ浏览器里自动全屏播放，这是腾讯浏览器的内置行为，不能修改，原因是由于腾讯浏览器挟持了video标签，由腾讯内置的播放器播放视频，但可以启用同层播放功能，可以解决视频覆盖Dom元素的问题，请参见同层播放。**
- **启用IE浏览器以最高级别的可用模式显示内容**
  小于IE10的浏览器需要启用最高级别的可用模式显示内容模式。示例如下：

```
<meta http-equiv="x-ua-compatible" content="IE=edge" >
```

- **如何设置自动播放时的自动全屏**
  给视频设置禁音，设置autoplay为true实现动播放，在监听ready事件调用 `fullscreenService.requestFullScreen` 实现全屏。示例如下：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="IE=edge" >
  <meta name="viewport" content="width=device-width, height=device-height, initial-sc
ale=1, maximum-scale=1, minimum-scale=1, user-scalable=no"/>
  <title>Aliplayer Functions</title>
  <link rel="stylesheet"
href="https://g.alicdn.com/de/prismplayer/2.9.17/skins/default/aliplayer-min.css" />
  <script type="text/javascript" charset="utf-8"
src="https://g.alicdn.com/de/prismplayer/2.9.17/aliplayer-min.js"></script>
  <!-- 请下载之后使用 -->
</head>
<body>
<div id="player-con"></div>
<script>
    var player = new Aliplayer({
        id: "player-con",
       source:"//example.aliyundoc.com/video/media02.mp4",
        width: "100%",
        height: "500px",
        autoplay: true,
        qualitySort: "asc",
        "mediaType": "video",
            preload: true,
        isLive: false,
    }, function (player) {
      player.mute()
        console.log("The player is created1");
    });
  // }
  player.on('ready',function(){
  player.fullscreenService.requestFullScreen()
  })
</script>
</body>
</html>
```

2021.12.31黄文飞提供，2022.1.2改用代码文本，屏蔽掉图片

- **iOS环境下，全屏播放视频时，无法正常使用弹幕** 2022.02.23根据用户反馈新增
  问题现象：iOS手机上直接使用播放器播放视频时，可以正常使用弹幕，但在全屏播放时，无法正常使用弹幕。
  问题原因：iOS下全屏，video会被原生UI接管，且处于最高层级，无法进行UI定制，也无法将弹幕元素置于video之上。
  解决方法：iOS下可以考虑通过模拟全屏的方式实现，即通过样式将播放器容器铺满整个屏幕，实现全屏播放的效果，同时可以正常使用弹幕功能。
- **如何在在uniapp onload生命周期引入SDK以及实例化播放器** 2022.03.07新增

```
 //在uniapp onload生命周期引入sdk以及实例化播放器
            onLoad() {
            const script = document.createElement('script');
            const link = document.createElement('link');
            link.href =
'https://g.alicdn.com/de/prismplayer/2.9.16/skins/default/aliplayer-min.css'
            link.setAttribute('rel','stylesheet')
            script.onload = () => {
              console.log('Aliplayer', window.Aliplayer);
              let player = new Aliplayer({
                "id": "player-con",
                source:"//player.alicdn.com/video/aliyunmedia.mp4",
                "width": "100%",
                "height": "500px",
                "autoplay": false,
                "isLive": false,
                "rePlay": false,
                "playsinline": true,
                "preload": true,
                "controlBarVisibility": "hover",
                "useH5Prism": true
              }, function (player) {
              })
            }
            script.src = 'https://g.alicdn.com/de/prismplayer/2.9.16/aliplayer-min.js
';
            script.setAttribute('type','text/javascript')
            document.body.appendChild(script);
            document.body.appendChild(link);
        }
```

- **player.seek（）方法在iOS下失效**
  需要优先在play事件和canplay事件中调用player.seek（）方法，否则可能会不生效。

```
 //优先在play事件和canplay事件调用seek，否则可能会不生效
     player.on('canplay',function(){
               player.seek(20)
             })
```

- **loadByUrl ios和android均无法使用**

```
seek会只能跳转，无法播放，
play在ios上只能从头播放
IOS上点全屏播放 video会被ios原生播放器劫持
    document.querySelector('.no1').onclick = function(){
              player.loadByUrl('//player.alicdn.com/resource/player/qupai.mp4')
        }
    优先监听 play 和 canplay 事件去 seek，有些浏览器可能不生效，可以考虑使用在第一次 timeupdate
的时候去 seek
     player.on('canplay',function(){
               player.seek(20)
              })
    ios下全屏被劫持，没有对应处理办法
```

- H5无法横屏这是为什么呢

播放器SDK没有横屏的接口，iOS横屏需要系统横屏，安卓全屏之后默认就是横屏

4.9. 参考文档

4.9.1. 实现自定义组件

Aliplayer2.3.0以上版本支持用户自定义组件，通过自定义组件用户可以在播放生命周期的某个节点插入自己的逻辑和实现自己的功能，比如弹幕、列表等。

生命周期节点

使用Aliplayer的关键方法名称及说明，如下表所示。

名称	说明
init	创建实例的时候调用，设置的初始参数在构建对象时，会传递给init方法。
createEl	创建组件的UI， 参数为播放器容器的div。
created	播放器创建完成，可以调用播放器的API。
ready	视频可播放状态。
play	开始播放。
pause	播放暂停。
playing	正在播放。
waiting	等待数据。
timeupdate	播放事件改变，通过第二个参数的timeStamp属性得到播放时间。
error	播放出错。
ended	播放结束。
dispose	播放器销毁。

自定义组件的实现

定义组件

有两种方法定义组件：

- 通过Aliplayer提供的Component方法

```
var StaticADComponent = Aliplayer.Component({
    init:function(adAddress,toAddress)
    {
      this.adAddress = adAddress;
      this.toAddress = toAddress;
      this.$html = $(html);
    },
    createEl:function(el)
    {
      this.$html.find('.ad').attr('src',this.adAddress);
      var $adWrapper = this.$html.find('.ad-wrapper');
      $adWrapper.attr('href',this.toAddress);
      $adWrapper.click(function(){
        Aliplayer.util.stopPropagation();
      });
      this.$html.find('.close').click(function(){
        this.$html.hide();
      });
      $(el).append(this.$html);
    },
    ready:function(player,e)
    {
    },
    play:function(player,e)
    {
       this.$html.hide();
    },
    pause:function(player,e)
    {
       this.$html.show();
    }
});
```

- **使用ES6的class类**

> **说明**
> 当您的项目是使用ES6的语法，通过webpack或者babel构建时，建议使用该方法。

```
export default class StaticADComponent
{
    constructor(adAddress,toAddress) {
      this.adAddress = adAddress;
      this.toAddress = toAddress;
      this.$html = $(html);
    }

    createEl(el)
    {
      this.$html.find('.ad').attr('src',this.adAddress);
      this.$html.attr('href',this.toAddress);
      let $adWrapper = this.$html.find('.ad-wrapper');
      $adWrapper.attr('href',this.toAddress);
      $adWrapper.click(()=>{
        Aliplayer.util.stopPropagation();
      });
      this.$html.find('.close').click(()=>{
        this.$html.hide();
      });
      $(el).append(this.$html);
    }


    ready(player,e)
    {
    }

    play(player,e)
    {
       this.$html.hide();
    }

    pause(player,e)
    {
       this.$html.show();
    }
}
```

设置components属性

定义好组件以后，需要注入播放器，让组件运行起来。设置组件提供3个属性，如下表所示。

名称	说明
name	组件名称，可通过名称得到组件。
type	组件类型。
args	组件的参数。

以下示例代码，所有参数均为一个组件的初始参数。

```
var player = new Aliplayer({
    id: "J_prismPlayer",
     autoplay: true,
     isLive:false,
     playsinline:true,
     controlBarVisibility:"click",
     cover:"",
     components:[
     {name:'adComponent',type:StaticAdComponent,args:
['http://example.aliyundoc.com/cover.png']},
     notParameComponent,
     {name:'adComponent1',type:notParameComponent}
     ]
    });
```

获取组件
提供getComponent方法获取实例组件，参数为组件的名称。

```
var component = player.getComponent('adComponent');
```

4.9.2. 配置延迟播放

Aliplayer2.1.0以上版本支持设置延迟播放和延迟提示信息，可用于边转码边播放的场景。

配置方式

延迟播放的提示信息会用到infoDisplay组件。
启用延迟播放时，需要把infoDisplay组件中的autoplay属性设置为true。

- **如果您有自定义skinLayout属性，则需要在skinLayout属性中启用infoDisplay组件。代码如下所示。**

```
skinLayout:[
    ......
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay", align: "cc"},
    ......
  ]
```

更多skinLayout属性信息，请参见skinLayout属性的配置**。**

- infoDisplay组件提供了两个属性，用于设置延迟播放的时间和提示。
  - autoPlayDelay：延迟多长时间后播放，单位为秒。
  - autoPlayDelayDisplayText：提示文本。例如“正在转码”，“请稍后”等。

4.9.3. 配置跨域访问

通过阅读本文，您可以了解H5播放、Flash播放和OSS播放如何配置跨域访问。

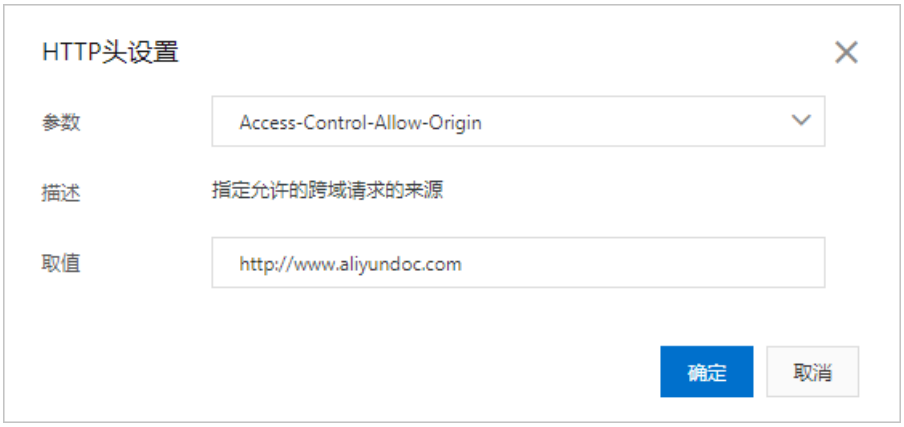
H5播放FLV、M3U8视频的跨域配置

当出现以下错误时，需要启用播放域名允许跨域访问。

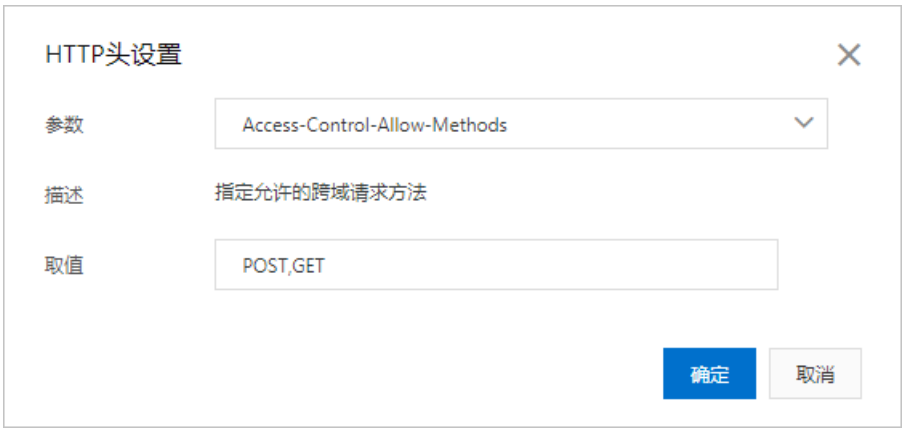
```
No ‘Access-Control-Allow-Origin’ header is present on the requested resource.
Origin ‘http://localhost:9030‘ is therefore not allowed access.
```

添加允许跨域访问的HTTP消息头。更多信息，请参见 配置HTTP消息头、设置跨域访问。

- Access-Control-Allow-Origin：指定允许的跨域请求的来源。取值为播放视频网站的域名，例如网站为https://www.aliyundoc.com，那么参数取值为 https://www.aliyundoc.com。示例图如下所示。



- Access-Control-Allow-Methods：指定允许的跨域请求方法。取值为POST或GET，如果您需要同时添加POST和GET，请使用半角逗号（,）隔开。示例图如下所示。



> **说明**
> 如果ts分片地址的域名和M3U8的地址的域名不一样，那么ts分片地址的域名也需要添加允许跨域访问的HTTP消息头。

Flash播放器的跨域配置

当出现以下的错误时，先排查域名是否备案，CNAME是否绑定。更多信息，请参见 域名准入标准、配置CNAME流程。

如果域名备案和CNAME绑定没有问题，那么就是跨域的问题。您可以添加crossdomain.xml访问策略文件，实现跨域配置。
将crossdomain.xml文件添加到视频地址域名的根目录下， 如果数据是存储在阿里云OSS上面， 在bucket的根目录下放置这个文件。
如果您使用的是阿里云点播服务，在开通服务时，点播会自动帮助你添加跨域文件。
crossdomain.xml文件如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<cross-domain-policy>
    <allow-access-from domain="*"/>
    <allow-http-request-headers-from domain="*" headers="*" secure="false"/>
</cross-domain-policy>
```

> **说明**
> 封面图片的地址与视频的地址不是同一个域名，封面图片所使用的域名也需要配置crossdomain.xml访问策略文件。

播放OSS存储视频的跨域设置

从播放器直接访问OSS需要开通Bucket的跨域资源共享。更多信息，请参见 控制台设置跨域访问、设置跨域资源共享。
创建跨域规则参数配置。

- **来源**：*。
- **允许 Methods**： 选择GET，POST，PUT，DELETE，HEAD。
- **允许 Headers**：*。
- **暴露Headers**：ETag。

> **注意**
> 请将该条CORS规则设置成所有CORS规则的第一条。

示例图如下所示。

4.9.4. 直播出错恢复处理

本文为您介绍在使用Aliplayer播放器进行直播时可能会遇到的问题及解决方式。

onM3u8Retry事件

- 事件定义
  在播放出错时，Aliyunplay播放器会重试5次取重新获取数据，同时会触发onM3u8Retry事件。通过订阅此事件，可以自定义显示消息，例如：主播暂时离开。
- 代码实现

```
player.on('onM3u8Retry',function(){
   console.log('主播暂时离开，请稍后.....');
 });
```

liveStreamStop事件

- 事件定义
  尝试数据恢复失败时，会触发liveStreamStop事件。通过订阅此事件，可以切换另一路流重新播放，或者自定义显示消息，例如：直播已结束。
- 代码实现
  - 切换另外一路可用流

```
player.on('liveStreamStop',function(){
   var newUrl = "新的直播流地址";
   player.loadByUrl(newUrl);
 });
```

  - 显示提示消息提示直播终止

```
player.on('liveStreamStop',function(){
   console.log('直播已结束');
 });
```

4.9.5. 配置skinLayout属性

通过阅读本文，您可以了解skinLayout的属性、设置规则和可定义属性。

skinLayout属性

Aliplayer可以通过skinLayout属性定制以下三个组件是否显示和显示的位置。

- 播放按钮
- Loading动画
- Controlbar的UI

您可以通过配置播放器的CSS文件：aliplayer-min.css，修改skinLayout属性。配置方式，请参见 设置播放器皮肤。

skinLayout设置规则

- skinLayout属性未设置，则默认显示全部。
- skinLayout设置为空集合（[]）或false时, 则全部隐藏。
- 如果是直播ControlBar的children属性，则只能设置为liveDisplay、fullScreenButton、subtitle、setting、volume和snapshot这几种UI组件。

可定义属性

align属性，组件相对于父级组件的对齐方式，可选项如下所示：

- `'cc'` ，相对于父组件绝对居中。
- `'tl'` ，相对于父组件左上对齐，受同级组件的占位影响，以组件的相对位置左上角作为偏移原点，可类比于css中的 `float: left` 。
- `'tr'` ，相对于父组件右上对齐，受同级组件的占位影响，以组件的相对位置右上角作为偏移原点，可类比于css中的 `float: right` 。

- `'tlabs'`，相对于父组件左上绝对对齐，不受同级组件的占位影响，以父组件左上角作为偏移原点。
- `'trabs'`，相对于父组件右上绝对对齐，不受同级组件的占位影响，以父组件右上角作为偏移原点。
- `'blabs'`，相对于父组件左下绝对对齐，不受同级组件的占位影响，以父组件左下角作为偏移原点。
- `'brabs'`，相对于父组件右下绝对对齐，不受同级组件的占位影响，以父组件右下角作为偏移原点。

x，y属性，组件相对于父级组件的位置，说明如下所示：

- x，{Number}，水平方向偏移量，偏移原点参考 `align` 的说明， `cc` 时无效。
- y，{Number}，垂直方向偏移量，偏移原点参考 `align` 的说明， `cc` 时无效。

点播默认配置

点播H5默认配置

```
skinLayout:[
   {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {
      name: "H5Loading", align: "cc"
    },
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay"},
    {name:"tooltip", align:"blabs",x: 0, y: 56},
    {name: "thumbnail"},
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
        {name: "progress", align: "blabs", x: 0, y: 44},
        {name: "playButton", align: "tl", x: 15, y: 12},
        {name: "timeDisplay", align: "tl", x: 10, y: 7},
        {name: "fullScreenButton", align: "tr", x: 10, y: 12},
        {name:"subtitle", align:"tr",x:15, y:12},
        {name:"setting", align:"tr",x:15, y:12},
        {name: "volume", align: "tr", x: 5, y: 10}
      ]
    }
  ]
```

点播Flash默认配置

```
skinLayout:[
    {name:"bigPlayButton", align:"blabs", x:30, y:80},
    {
      name:"controlBar", align:"blabs", x:0, y:0,
      children: [
        {name:"progress", align:"tlabs", x: 0, y:0},
        {name:"playButton", align:"tl", x:15, y:26},
        {name:"nextButton", align:"tl", x:10, y:26},
        {name:"timeDisplay", align:"tl", x:10, y:24},
        {name:"fullScreenButton", align:"tr", x:10, y:25},
        {name:"streamButton", align:"tr", x:10, y:23},
        {name:"volume", align:"tr", x:10, y:25}
      ]
    },
    {
      name:"fullControlBar", align:"tlabs", x:0, y:0,
      children: [
        {name:"fullTitle", align:"tl", x:25, y:6},
        {name:"fullNormalScreenButton", align:"tr", x:24, y:13},
        {name:"fullTimeDisplay", align:"tr", x:10, y:12},
        {name:"fullZoom", align:"cc"}
      ]
    }
]
```

配置效果图

直播默认配置

直播H5模式默认配置

```
skinLayout: [
    {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay", align: "cc"},
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
          {name:"liveDisplay", align:"tlabs", x: 15, y:6},
          {name:"fullScreenButton", align:"tr", x:10, y: 10},
          {name:"subtitle", align:"tr",x:15, y:12},
          {name:"setting", align:"tr",x:15, y:12},
          {name:"volume", align:"tr", x:5, y:10}
        ]
    }
  ]
```

直播Flash模式默认配置

```
skinLayout: [
    {name: "bigPlayButton", align: "blabs", x: 30, y: 80},
    {name: "errorDisplay", align: "tlabs", x: 0, y: 0},
    {name: "infoDisplay", align: "cc"},
    {
      name: "controlBar", align: "blabs", x: 0, y: 0,
      children: [
          {name:"liveDisplay", align:"tlabs", x: 15, y:25},
          {name:"fullScreenButton", align:"tr",  x:10, y:25},
          {name:"volume", align:"tr",  x:10, y:25}
        ]
    }
  ]
```

配置效果图

5.Android播放器

5.1. 运行Demo源码

Demo源码提供了播放器的常用功能示例，集成播放器SDK前，可以先运行Demo源码以了解和体验播放器SDK的功能。您也可以不单独集成SDK，直接集成完整的Demo源码来直接使用播放器SDK，或根据需要集成Demo源码中的部分模块来使用。

体验Demo

体验Demo提供了完整的产品级的交互UI和业务源码，包含短视频、播放器和上传等SDK。您可以扫描二维码下载Demo App体验阿里云播放器相关功能。 二维码地址 请参见Demo体验。同时阿里云也免费提供Android播放器SDK Demo源码。 Android播放器SDK Demo源码中 内置了UI，以下分别介绍使用和不使用内置的UI来集成播放器Demo源码的场景及操作步骤 。

场景	说明
运行完整Demo（含内置UI ）	适用于需要使用内置UI，且想要使用Demo源码中提供的所有功能。 此场景下无需单独集成SDK，直接导入完整的Demo源码并运行即可。
集成Demo模块（含内置UI）	适用于需要使用内置UI，但只想使用Demo源码中的部分功能。 此场景下无需单独集成SDK，只需选择性的导入Demo源码中的功能模块运行即可。
集成Demo（不含内置UI）	适用于不需要内置UI（自定义UI或不使用UI），又想省去自己开发的麻烦，直接使用Demo中封装好的功能。 此场景下需要先集成SDK，再集成Demo中的相关文件。

环境要求

类别	说明
系统版本	支持Android 4.3及以上版本。
手机芯片	架构要求： • armv7。 • arm64。
开发工具	推荐使用Android Studio，本文操作步骤基于Android Studio开发。下载地址：Android Studio 。

前提条件

下载Android播放器SDK包（包含了播放器SDK及Demo源码），推荐下载使用最新版本。下载地址请参见 SDK简介与下载。

解压后的目录结构如下：

SDK目录结构

文件名	作用
demo	播放器的Demo源码。Demo源码的目录结构参见 demo目录结构。
JavaDoc	播放器API文档。

文件名	作用
sdk	播放器SDK的aar库。
X.X.XReleaseNote	版本说明。

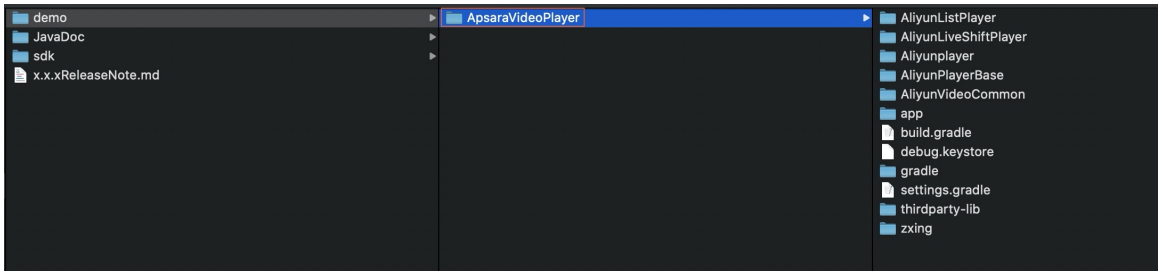
demo目录结构

模块名	模块作用
AliyunListPlayer	列表播放模块，对应列表播放器的示例代码。
AliyunLiveShiftPlayer	直播时移模块，对应直播时移的示例代码。
AliyunPlayer	播放器模块，对应播放器的示例代码。
AliyunPlayerBase	播放器Demo的Base模块，使用gradle方式集成了播放器SDK。 说明 此模块必须导入，无论导入Demo中的哪一个和播放器相关的模块，都需要导入此模块。
AliyunVideoCommon	阿里云项目公共模块，包括一些工具类。 说明 此模块必须导入，无论导入Demo中的哪一个和播放器相关的模块，都需要导入此模块。
zxing	二维码扫描模块。
thirdparty-lib	包含Demo中的所有的依赖。 说明 此模块必须导入，但thirdparty-lib并不是AndroidStudio中的Module项目，需要手动拷贝到项目中，和AliyunPlayerBase、AliyunVideoCommon等文件夹目录平级即可。

运行完整Demo（含内置UI）

通过导入完整的播放器SDK Demo包，可不再单独集成SDK，直接开始使用包含了内置UI的播放器功能。

1. 在Android Studio的导航栏选择 **File > Open**，在弹框中选择demo中的ApsaraVideoPlayer并导入。



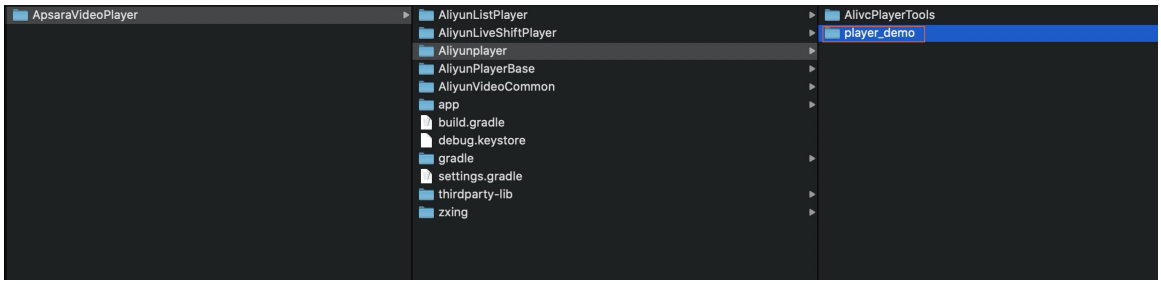
集成Demo模块（含内置UI）

通过选择导入播放器SDK Demo源码中的部分模块，可不集成SDK，选择性的使用封装好的包含内置UI播放器的部分功能。

> **说明** 以下步骤以集成AliyunPlayer模块为例，介绍如何快速集成Demo中的模块。

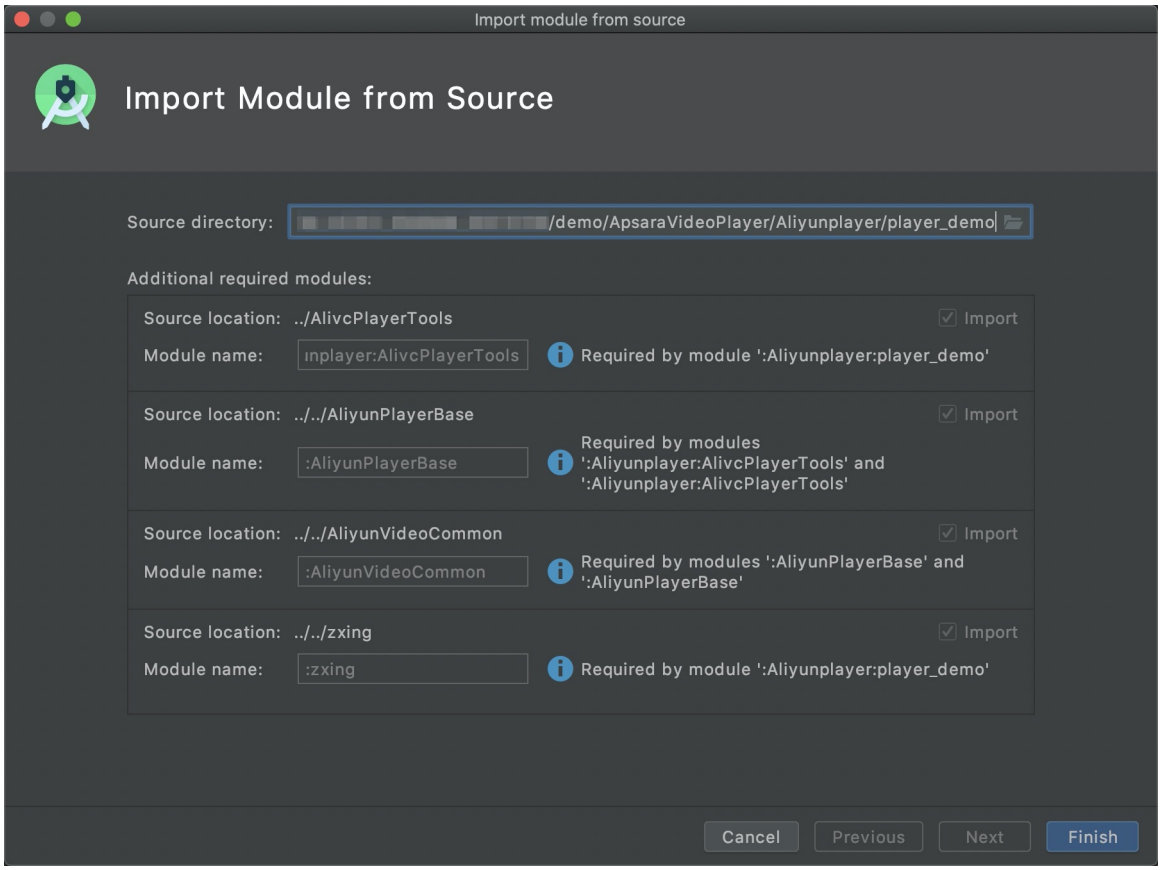
1. 单击Android Studio的导航栏，选择**File > New > Import Module...**，选择需要导入的模块*player_demo*。

   > **说明** 其中AliyunPlayerBase、AliyunVideoCommon和thirdparty-lib模块必须导入，其余模块根据需要选择性导入。详细请参见 demo目录结构 。

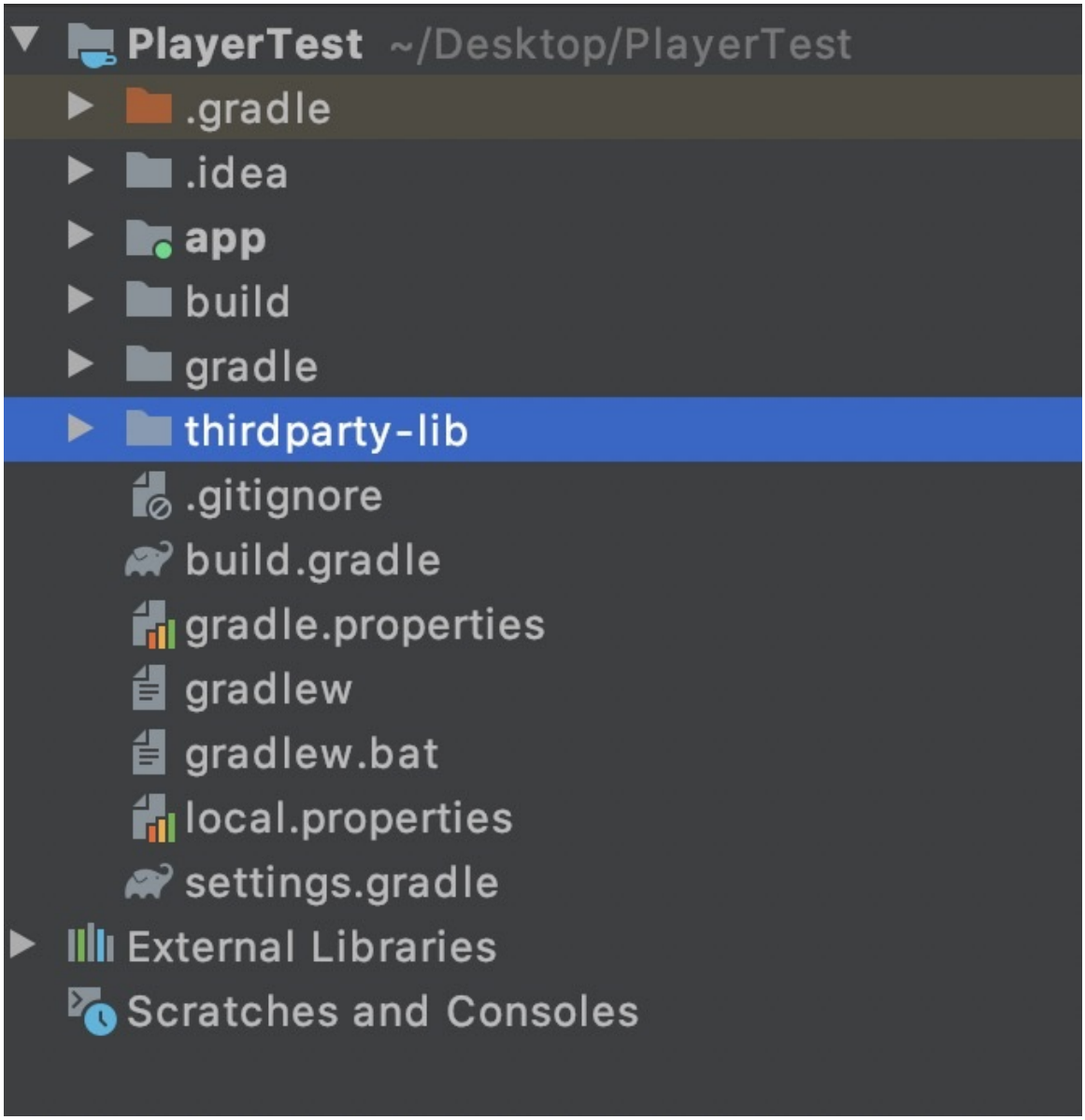
   2021.10.29 开发韩宇提供

2. 单击**Finish**，AndroidStudio会自动选中其他需要导入的模块。

   2021.10.29 开发韩宇提供

3. 导入成功后，需要手动引入thirdparty-lib，将该文件夹拷贝到项目中，并修改Project的build.gradle文件，引用thirdparty-lib文件和Maven仓库。



2021.10.29 开发韩宇提供

```
buildscript { apply from: 'thirdparty-lib/config.gradle' repositories { google() jc
enter() maven { url "http://maven.aliyun.com/nexus/content/repositories/releases" }
//引入投屏需要使用的maven仓库 maven { url 'http://4thline.org/m2' } } } allprojects { r
epositories { google() jcenter() maven { url
"http://maven.aliyun.com/nexus/content/repositories/releases" } //引入投屏需要使用的ma
ven仓库 maven { url 'http://4thline.org/m2' } } }
```

4. **进入app目录下的build.gradle文件，修改参数compileSdkVersion、buildToolsVersion、minSdkVersion、targetSdkVersion的值，使其与播放器SDK Demo中Module目录下thirdparty-lib/config.gradle文件中的值保持一致。**

```
android {
    compileSdkVersion externalCompileSdkVersion
    buildToolsVersion externalBuildToolsVersion

    defaultConfig {
        applicationId "[redacted]"
        minSdkVersion externalLiveMinSdkVersion
        targetSdkVersion externalTargetSdkVersion
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
```

2021.10.29 开发韩宇提供

如果出现下图中的报错，请参考 故障排除 排除故障。

```
Execution failed for task ':app:processDebugMainManifest'.
> Manifest merger failed : Attribute application@theme value=(@style/Theme.Test) from AndroidManifest.xml:14:9-42
    is also present at [:zxing] AndroidManifest.xml:19:9-40 value=(@style/AppTheme).
    Suggestion: add 'tools:replace="android:theme"' to <application> element at AndroidManifest.xml:8:5-22:19 to
  override.
```

2021.10.29 开发韩宇提供

5. 在app目录下的build.gradle文件中，删除AndroidX相关的依赖，并增加如下依赖。

```
implementation externalAndroidSupportV4 implementation externalAndroidAppCompatV7 i
mplementation project(':Aliyunplayer:player_demo')
```

集成完后，可通过如下代码跳转到播放器的AliyunPlayer模块。

```
Intent intent = new Intent(MainActivity.this,AliyunPlayerSettingActivity.class); st
artActivity(intent);
```

集成Demo（不含内置UI）

播放器Demo中包含许多内置UI，如果您不需要内置UI（自定义UI或不使用UI），又想省去自己开发的麻烦，直接使用Demo中封装好的功能，则可以通过以下步骤实现。

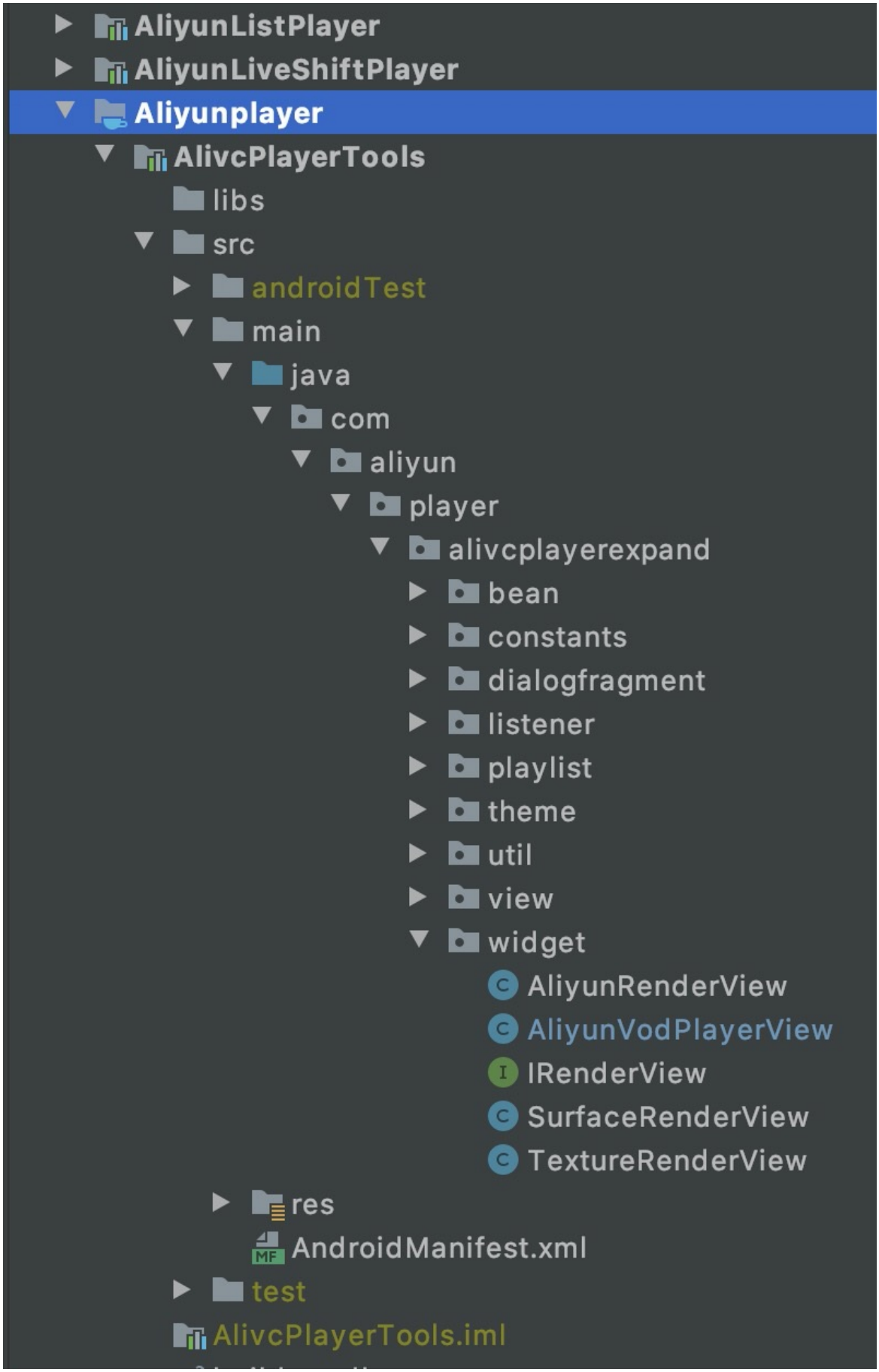
1. 集成播放器SDK。

   请参见 快速集成。

2. 拷贝Demo中部分代码。

   拷贝 *Aliyunplayer/AlivcPlayerTools/src/main/java/com/aliyun/player/alivecpalyerexpand/widget*目录下的文件：AliyunRenderView、IRenderView、TextureRenderView、SurfaceRenderView。

   > **说明**
   >
   > - AliyunRenderView为播放器的封装，将播放器和渲染的View绑定。
   > - IRenderView用于统一渲染接口，SurfaceRenderView和TextureRenderView 都实现了IRenderView。
   > - SurfaceRenerView和TextureRenderView分别对应使用SurfaceView和TextureView来渲染视频画面。
   > - AliyunVodPlayerView无需拷贝，该文件提供的是AliyunRenderView的使用示例代码。


AliyunListPlayer
AliyunLiveShiftPlayer
Aliyunplayer
AlivcPlayerTools
libs
src
androidTest
main
java
com
aliyun
player
alivcplayerexpand
bean
constants
dialogfragment
listener
playlist
theme
util
view
widget
AliyunRenderView
AliyunVodPlayerView
IRenderView
SurfaceRenderView
TextureRenderView
res
AndroidManifest.xml
test
AlivcPlayerTools.iml

3. 使用AliyunRenderView。

   可参考AliyunVodPlayerView文件中的示例代码。

   i. 创建AliyunRenderView。

```
<?xml version="1.0" encoding="utf-8"?> <!-- LinearLayout是根布局，可以修改为其他ViewGroup，这里仅是示例代码 --> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent" android:layout_height="match_parent"> <!-- 播放器自定义View，封装了播放器和渲染View，属性说明： android:id为ViewId，请根据实际填写 android:layout_width为View的宽度，可修改，会影响播放画面 android:layout_height为View的高度，可修改，会影响播放画面 --> <com.aliyun.playertest.playerDemo.AliyunRenderView android:id="@+id/aliyunRenderView" android:layout_width="match_parent" android:layout_height="200dp" /> </LinearLayout>
```

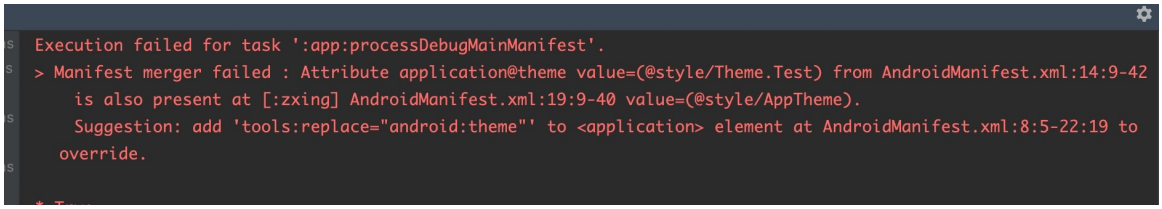
   ii. 调用接口。

```
java //设置渲染View的类型，可选 SurfaceType.TEXTURE_VIEW 和 SurfaceType.SURFACE_VIEW aliyunRenderView.setSurfaceType(AliyunRenderView.SurfaceType.SURFACE_VIEW); //设置监听 aliyunRenderView.setXXXListener; //设置播放源，setDataSource为重载方法，还可以设置 sts，playAuth 等数据源 aliyunRenderView.setDataSource(urlSource); //播放相关 aliyunRenderView.prepare(); aliyunRenderView.start() aliyunRenderView.pause() aliyunRenderView.stop() aliyunRenderView.release()
```

> **说明** 上述代码展示了部分接口，其余部分的接口可以参照AliyunVodPlayerView文件的示例代码来使用，也可以根据播放器SDK的API接口文档来使用AliyunRenderView。

故障排除

使用内置UI集成部分播放器Demo时，当出现下图中报错时，请根据下述步骤排除故障。



1. 打开*app/AndroidManifest.xml*文件，在 application 标签中添加如下代码。

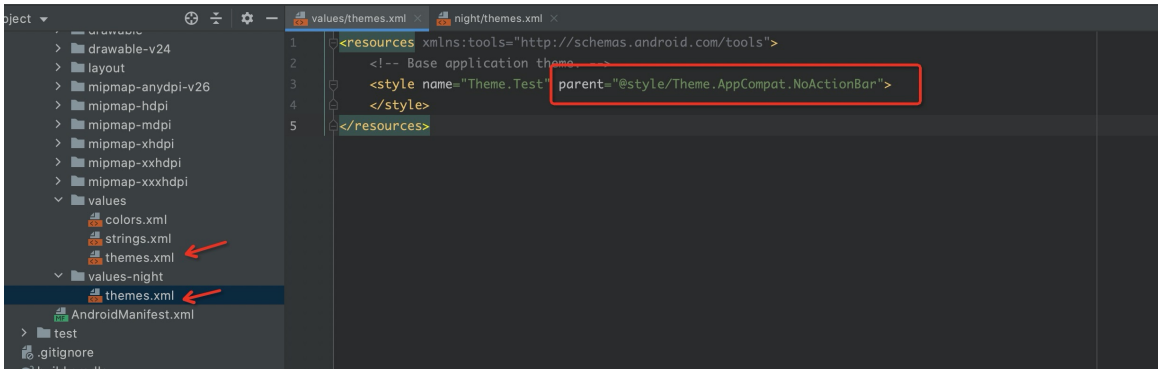
```
android:allowBackup="true"
```

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Test"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Test">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

2021.10.29 开发韩宇提供

2. 打开*values/themes.xml*及*values-night/themes.xml*文件，修改 android:theme 的themes资源文件为*@style/Theme.AppCompat.NoActionBar*。



2021.10.29 开发韩宇提供

5.2. 快速集成

本文介绍Android播放器SDK的环境要求、集成方式及使用参考。

环境要求

类别	说明
系统版本	支持Android 4.3及以上版本。
手机芯片	架构要求： • armv7。 • arm64。
开发工具	推荐使用Android Studio，本文操作步骤基于Android Studio开发。下载地址：Android Studio 。

前提条件

本地集成SDK时，需要先下载Android播放器SDK包 （包含了播放器SDK及Demo源码），推荐下载使用最新版本。 下载地址请参见 SDK简介与下载 。

解压后的目录结构如下：

SDK包目录结构

文件名	作用
demo	播放器的Demo源码。
JavaDoc	播放器API文档。
sdk	播放器SDK的aar库。sdk的目录结构参见 sdk目录结构。
X.X.XReleaseNote	版本说明。

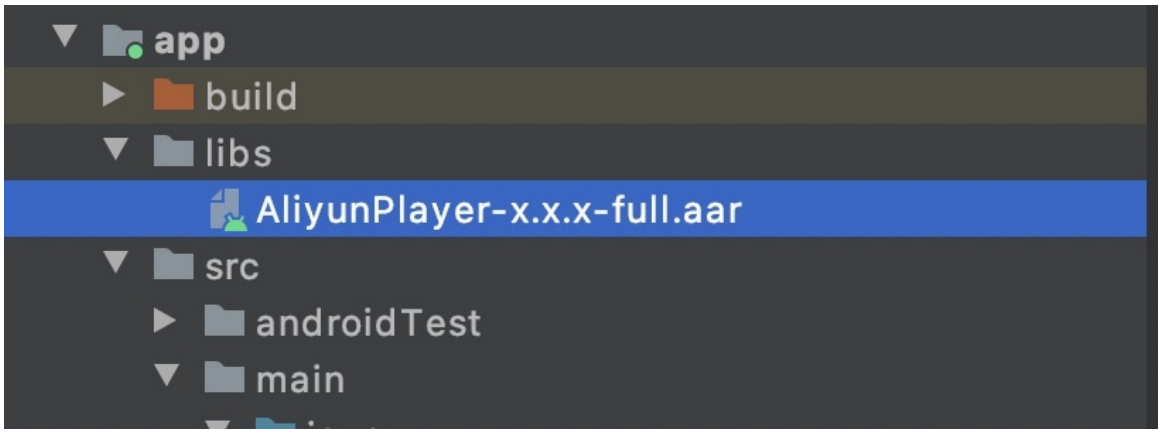
sdk目录结构

文件名	说明
AliyunPlayer-x.x.x-full.aar	完整的aar包，包含了FFmpeg动态库的包。
AliyunPlayer-x.x.x-part.aar	aar包，不包含FFmpeg的动态库的包。
AlivcArtp-x.x.x.aar	支持artp协议，非必须。
AlivcArtc-x.x.x.aar	支持artc协议，非必须。

> 说明
> - 如果不集成短视频SDK，直接依赖AliyunPlayer-x.xx.x-full.aar包即可。
> - 如果集成播放器的同时，也集成短视频SDK，那么播放器的SDK依赖AliyunPlayer-x.x.x-part包，同时使用共通的FFmpeg版本，即需要额外依赖com.aliyun.video.android:AlivcFFmpeg:x.x.x这个包。
> - 如果集成时使用了错误的SDK包，会导致FFmpeg冲突。

SDK集成（本地集成）

1. 拷贝需要的aar包到工程的libs目录下（如果没有libs文件夹，手动创建libs文件夹即可）。



2. 修改Project下的build.gradle文件，在 allprojects 的repositories节点中增加flatDir的设置，示例如下：

```
flatDir { dirs 'libs' }
```

3. 修改App的build.gradle文件，dependencies节点中增加对aar的引用和Conan的引用，示例如下：

```
dependencies { implementation fileTree(dir: 'libs', include: ['*.aar']) //阿里云播放器5.3.0 版本以前需要引入AlivcConan库，5.3.0及其以后的版本不需要引入。 implementation 'com.alivc.conan:AlivcConan:x.x.x' }
```

4. **混淆配置。**

   在App的Proguard-rules.pro文件中添加以下混淆配置，示例如下：

```
-keep class com.alivc.**{*;} -keep class com.aliyun.**{*;} -keep class com.cicada.**{*;} -dontwarn com.alivc.** -dontwarn com.aliyun.** -dontwarn com.cicada.**
```

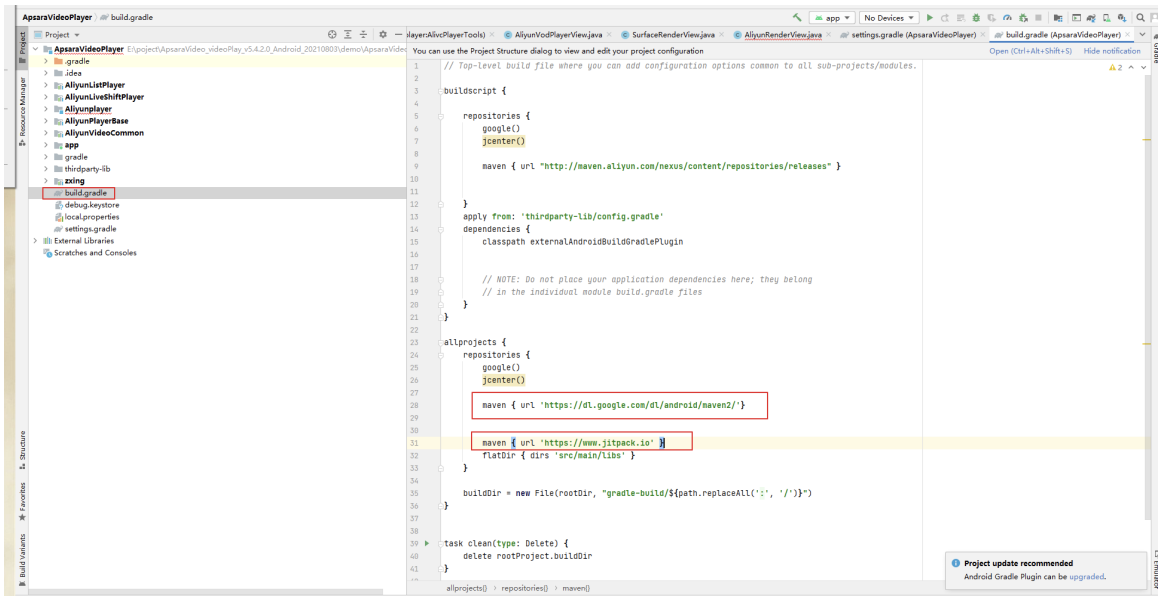
SDK集成（Gradle集成）

> **说明** 请确保网络正常，并且可以正常访问阿里云Maven仓库。如果由于网络故障，无法从Maven仓库下载到播放器SDK包，则建议通过本地集成方式来集成SDK，详细请参见 SDK集成（本地集成）。

1. **在Project的build.gradle中增加阿里云的Maven地址依赖。示例如下：**

```
maven { url "http://maven.aliyun.com/nexus/content/repositories/releases" }
```

   操作后的截图如下：



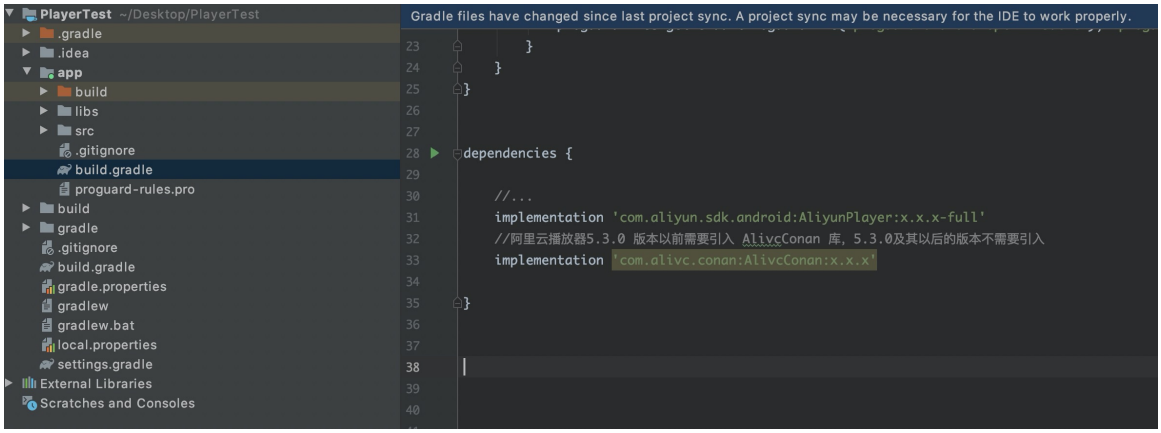
   2021.10.29测试远春提供

2. **修改App的build.gradle文件，dependencies节点中增加依赖。示例如下：**

> **说明** 请确保引入的播放器SDK版本号填写正确，否则将报错并引入失败。

```
implementation 'com.aliyun.sdk.android:AliyunPlayer:x.x.x-full' //阿里云播放器5.3.0 版本以前需要引入AlivcConan库，5.3.0及其以后的版本不需要引入 implementation 'com.alivc.conan:AlivcConan:x.x.x'
```

   操作后的截图如下：



常见集成问题
播放器问题

功能使用文档
基础功能

进阶功能

5.3. 基础功能
本文提供Android播放器基础功能的使用示例。

创建播放器
本节介绍如何用最简单的方式让Android播放器SDK播放视频，按照播放方式的不同可以分为手动播放和自动播放。

1. 创建播放器。

   通过 `AliPlayerFactory` 类创建AliPlayer播放器。

```
AliPlayer aliPlayer = AliPlayerFactory.createAliPlayer(context);
aliPlayer.setTraceId("traceId");  //traceId为设备或用户的唯一标识符，通常为imei或idfa
```

   2021.12.30：增加traceid透传示例代码

2. 设置监听器。

   播放器支持设置多个监听器。

   - `OnPreparedListener` 必须设置，因为手动播放需要在 `OnPreparedListener` 回调中调用 `aliPlayer.start()` 开始播放。
   - `OnErrorListener` 、 `OnCompletionListener` 、 `OnLoadingStatusListener` 、 `OnInfoListener` 较为重要，建议您设置。

```
aliPlayer.setOnErrorListener(new IPlayer.OnErrorListener() {
    //此回调会在使用播放器的过程中，出现了任何错误，都会回调此接口。
    @Override
    public void onError(ErrorInfo errorInfo) {
        ErrorCode errorCode = errorInfo.getCode(); //错误码
        String errorMsg =errorInfo.getMsg(); //错误描述
        //出错后需要停止掉播放器
        aliPlayer.stop();
    }
});
aliPlayer.setOnPreparedListener(new IPlayer.OnPreparedListener() {
    // 调用aliPlayer.prepare()方法后，播放器开始读取并解析数据。成功后，会回调此接口。
```

```
    @Override
    public void onPrepared() {
        //一般调用start开始播放视频
        aliPlayer.start();
    }
});
aliPlayer.setOnCompletionListener(new IPlayer.OnCompletionListener() {
    //播放完成之后，就会回调到此接口。
    @Override
    public void onCompletion() {
        //一般调用stop停止播放视频
        aliPlayer.stop();
    }
});
aliPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    //播放器中的一些信息，包括：当前进度、缓存位置等等
    @Override
    public void onInfo(InfoBean infoBean) {
        InfoCode code = infoBean.getCode(); //信息码
        String msg = infoBean.getExtraMsg();//信息内容
        long value = infoBean.getExtraValue(); //信息值
        //当前进度：InfoCode.CurrentPosition
        //当前缓存位置：InfoCode.BufferedPosition
    }
});
aliPlayer.setOnLoadingStatusListener(new IPlayer.OnLoadingStatusListener() {
    //播放器的加载状态， 网络不佳时，用于展示加载画面。
    @Override
    public void onLoadingBegin() {
        //开始加载。画面和声音不足以播放。
        //一般在此处显示圆形加载
    }
    @Override
    public void onLoadingProgress(int percent, float netSpeed) {
        //加载进度。百分比和网速。
    }
    @Override
    public void onLoadingEnd() {
        //结束加载。画面和声音可以播放。
        //一般在此处隐藏圆形加载
    }
});
```

3. 创建DataSource。
   - Android播放器SDK支持5种点播播放方式，包括：UrlSource播放、VidAuth播放（视频点播用户推荐使用）、VidSts播放、VidMps播放、加密播放。
   - Android播放器SDK支持2种直播播放方式，UrlSource播放和加密播放。

> **说明**
>
> - UrlSource是直接通过URL播放，其余的三种是通过vid进行播放：VidSts，VidAuth（推荐）限点播用户使用；VidMps仅限媒体处理（MPS）用户使用。
> - 接入地域Region的设置，请参见 **点播地域标识** 。
> - MPS视频播放的流程与概念，请参见 **视频播放** 。

点播视频播放
中国站

点播UrlSource播放 | VidAuth播放 | 点播VidSts播放 | VidMps播放 | 点播加密播放

直播视频播放
中国站

直播UrlSource播放 | 直播DRM加密播放

4. **设置显示View。**

   播放器支持SurfaceView和TextureView，任选其中一种即可。

   - **设置SurfaceView，示例如下：**

```
SurfaceView surfaceView = findViewById(R.id.surface_view);
surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        aliPlayer.setSurface(holder.getSurface());
    }
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int h
eight) {
        aliPlayer.surfaceChanged();
    }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        aliPlayer.setSurface(null);
    }
});
```

   - **设置TextureView，示例如下：**

```
TextureView textureView = findViewById(R.id.texture_view);
textureView.setSurfaceTextureListener(new TextureView.SurfaceTextureListener() {
    @Override
    public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int
height) {
        aliPlayer.setSurface(new Surface(surface));
    }
    @Override
    public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width, in
t height) {
        aliPlayer.surfaceChanged();
    }
    @Override
    public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
        aliPlayer.setSurface(null);
        return false;
    }
    @Override
    public void onSurfaceTextureUpdated(SurfaceTexture surface) {
    }
});
```

5. （可选）开启自动播放，默认为关闭状态。

```
aliPlayer.setAutoPlay(true);
```

6. 准备播放。

   调用 `aliPlayer.prepare()` 开始读取并解析数据。

```
aliPlayer.prepare();
```

7. 开始播放。
   - 如果未开启自动播放，需要在 `OnPrepard` 回调中调用 `aliPlayer.start()` 开始播放视频。
   - 如果开启了自动播放，则不需要调用 `aliPlayer.start()` ，数据解析完成后将开始自动播放视频。

```
aliPlayer.start();//开始之后可以调用pause()暂停播放视频。
```

控制播放

Android播放器SDK支持从指定时间点播放、开始、暂停、停止播放等操作。 指定时间播放即seek。

开始播放

指开始播放视频，由 `start` 接口实现。示例如下：

```
aliyunVodPlayer.start();
```

从指定时间开始播放

指跳转到某个时刻进行播放，由 `seekTo` 接口实现。适用于用户拖拽进度条，或续播等需要从指定时间点开始播放的场景。示例如下：

```
//posotion为指定的时间，单位：秒。
aliyunVodPlayer.seekTo(long position);
```

暂停播放

指暂停播放视频，由 `pause` 接口实现。示例如下：

```
aliyunVodPlayer.pause();
```

停止播放

指停止播放视频，由 `stop` 接口实现。示例如下：

```
aliyunVodPlayer.stop();
```

设置显示模式

Android播放器SDK支持填充、旋转、镜像等显示设置。

填充

支持设置宽高比适应、宽高比填充和拉伸填充这3种画面填充模式，由 `setScaleMode` 接口实现。示例如下：

```
//设置宽高比适应（将按照视频宽高比等比缩小到view内部，不会有画面变形）
aliyunVodPlayer.setScaleMode(ScaleMode.SCALE_ASPECT_FIT);
//设置宽高比填充（将按照视频宽高比等比放大，充满view，不会有画面变形）
aliyunVodPlayer.setScaleMode(ScaleMode.SCALE_ASPECT_FILL);
//设置拉伸填充（如果视频宽高比例与view比例不一致，会导致画面变形）
aliyunVodPlayer.setScaleMode(ScaleMode.SCALE_TO_FILL);
```

旋转

指画面按指定角度旋转，由 `setRotateMode` 接口实现。设置后还可查询旋转角度。示例如下：

```
//设置画面顺时针旋转0度
aliyunVodPlayer.setRotateMode(RotateMode.ROTATE_0);
//设置画面顺时针旋转90度
aliyunVodPlayer.setRotateMode(RotateMode.ROTATE_90);
//设置画面顺时针旋转180度
aliyunVodPlayer.setRotateMode(RotateMode.ROTATE_180);
//设置画面顺时针旋转270度
aliyunVodPlayer.setRotateMode(RotateMode.ROTATE_270);
//获取旋转角度
aliyunVodPlayer.getRotateMode();
```

镜像

支持水平镜像、垂直镜像和无镜像，由 `setMirrorMode` 接口实现。示例如下：

```
//设置无镜像
aliyunVodPlayer.setMirrorMode(MirrorMode.MIRROR_MODE_NONE);
//设置水平镜像
aliyunVodPlayer.setMirrorMode(MirrorMode.MIRROR_MODE_HORIZONTAL);
//设置垂直镜像
aliyunVodPlayer.setMirrorMode(MirrorMode.MIRROR_MODE_VERTICAL);
```

获取播放信息

Android播放器SDK支持获取当前的播放进度、播放时长和缓冲进度信息。

获取当前播放进度

指获取当前的播放时刻，需要在onInfo回调中获取，由 `getExtraValue` 接口实现。示例如下：

```
mAliPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if(infoBean.getCode() == InfoCode.CurrentPosition){
            //extraValue为当前播放进度，单位为毫秒
            long extraValue = infoBean.getExtraValue();
        }
    }
});
```

获取播放时长

指获取视频总时长。需要在视频加载完成以后才可以获取到，可以在onPrepared事件后获取。
由 `getDuration` 接口实现。示例如下：

```
long duration = mAliPlayer.getDuration();
```

获取缓冲进度

指获取视频当前的缓冲进度，需要在onInfo回调中获取，由 `getExtraValue` 接口实现。示例如下：

```
mAliPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if(infoBean.getCode() == InfoCode.BufferedPosition){
            ////extraValue为当前缓冲进度，单位为毫秒
            long extraValue = infoBean.getExtraValue();
        }
    }
});
```

监听播放状态

指监听播放器的状态，onStateChanged回调参数为当前播放器状态。示例如下：

```
mAliPlayer.setOnStateChangedListener(new IPlayer.OnStateChangedListener() {
    @Override
    public void onStateChanged(int newState) {
        /*
          int idle = 0;
          int initalized = 1;
          int prepared = 2;
          int started = 3;
          int paused = 4;
          int stopped = 5;
          int completion = 6;
          int error = 7;
      */
    }
});
```

设置音量

设置音量包括音量调节和静音设置。

音量调节

指调节音量大小，由 `setVolume` 接口实现。设置后还可获取音量信息。示例如下：

```
//volume的值为0～1之间的实数。
aliyunVodPlayer.setVolume(1f);
//获取音量信息。
aliyunVodPlayer.getVolume();
```

静音设置

指将播放中的视频设置为静音状态，由 `setMute` 接口实现。示例如下：

```
aliyunVodPlayer.setMute(true);
```

倍速播放

Android播放器SDK提供了倍速播放视频的功能，通过设置 `setSpeed` 方法，能够以0.5倍~5倍速去播放视频。同时保持变声不变调。示例如下：

```
//设置倍速播放：支持0.5～5倍速的播放，通常按0.5的倍数来设置，例如0.5倍、1倍、1.5倍等
aliyunVodPlayer.setSpeed(1.0f);
```

多清晰度设置

如果使用VID方式（VidAuth及VidSts）播放，无需额外设置。Android播放器SDK会从点播服务获取清晰度列表。Android播放器SDK支持获取和切换清晰度，UrlSource方式暂不支持此设置。

获取清晰度

当视频加载完成后，可以获取视频的清晰度。

```
//获取媒体所有的流信息
List<TrackInfo> trackInfos = mAliPlayer.getMediaInfo().getTrackInfos();
//遍历并获取清晰度
for (TrackInfo trackInfo : trackInfos) {
     if(trackInfo.getType() == TrackInfo.Type.TYPE_VOD){
             //获取视频清晰度
        String vodDefinition = trackInfo.getVodDefinition();
     }
}
```

切换清晰度

通过 `selectTrack` 方法切换清晰度，传递对应TrackInfo的index即可。

```
mAliPlayer.selectTrack(index);
```

清晰度切换通知

清晰度切换成功与失败回调。

```
mAliPlayer.setOnTrackChangedListener(new IPlayer.OnTrackChangedListener() {
    @Override
    public void onChangedSuccess(TrackInfo trackInfo) { }
    @Override
    public void onChangedFail(TrackInfo trackInfo, ErrorInfo errorInfo) { }
});
```

循环播放

Android播放器SDK提供了循环播放视频的功能。调用 `setLoop` 开启循环播放，播放完成后，将会自动从头开始播放视频。示例如下：

```
aliyunVodPlayer.setLoop(true);
```

同时循环开始的回调将会使用 `onInfo` 中通知。示例如下：

```
aliyunVodPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if (infoBean.getCode() == InfoCode.LoopingStart){
            //循环播放开始事件。
        }
    }
});
```

相关文档

- 接口说明
- 进阶功能

5.4. 进阶功能

本文提供Android播放器进阶功能的使用示例。

播放
短视频列表播放

针对典型的短视频列表播放场景，iOS播放器SDK提供了完善的列表播放功能，结合预加载等机制大幅改善短视频的起播速度。长视频场景不建议使用该功能。

1. 创建播放器。

   通过 `AliPlayerFactory` 类创建AliListPlayer播放器。示例如下：

```
AliListPlayer aliyunListPlayer;
.....
aliyunListPlayer = AliPlayerFactory.createAliListPlayer(getApplicationContext());
aliyunListPlayer.setTraceId("traceId");  //traceId为设备或用户的唯一标识符，通常为imei或idfa
```

   2021.12.30：增加traceid透传示例代码

2. （可选）设置监听器。

   创建列表播放时，监听器为非必须配置，但如果不设置，将无法收到播放器的播放失败、播放进度等事件通知，因此，建议您设置。其中 `OnPreparedListener` 、 `OnErrorListener` 、 `OnCompletionListener` 、 `OnLoadingStatusListener` 、 `OnInfoListener` 较为重要，建议您设置。

```
aliyunListPlayer.setOnCompletionListener(new IPlayer.OnCompletionListener() {
    @Override
    public void onCompletion() {
        //播放完成事件
    }
});
aliyunListPlayer.setOnErrorListener(new IPlayer.OnErrorListener() {
    @Override
    public void onError(ErrorInfo errorInfo) {
        //出错事件
    }
});
aliyunListPlayer.setOnPreparedListener(new IPlayer.OnPreparedListener() {
    @Override
    public void onPrepared() {
        //准备成功事件
    }
});
aliyunListPlayer.setOnVideoSizeChangedListener(new
IPlayer.OnVideoSizeChangedListener() {
    @Override
    public void onVideoSizeChanged(int width, int height) {
        //视频分辨率变化回调
    }
});
aliyunListPlayer.setOnRenderingStartListener(new IPlayer.OnRenderingStartListener()
{
    @Override
    public void onRenderingStart() {
        //首帧渲染显示事件
    }
});
```

```
aliyunListPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(int type, long extra) {
        //其他信息的事件，type包括了：循环播放开始，缓冲位置，当前播放位置，自动播放开始等
    }
});
aliyunListPlayer.setOnLoadingStatusListener(new IPlayer.OnLoadingStatusListener() {
    @Override
    public void onLoadingBegin() {
        //缓冲开始。
    }
    @Override
    public void onLoadingProgress(int percent, float kbps) {
        //缓冲进度
    }
    @Override
    public void onLoadingEnd() {
        //缓冲结束
    }
});
aliyunListPlayer.setOnSeekCompleteListener(new IPlayer.OnSeekCompleteListener() {
    @Override
    public void onSeekComplete() {
        //拖动结束
    }
});
aliyunListPlayer.setOnSubtitleDisplayListener(new
IPlayer.OnSubtitleDisplayListener() {
    @Override
    public void onSubtitleShow(long id, String data) {
        //显示字幕
    }
    @Override
    public void onSubtitleHide(long id) {
        //隐藏字幕
    }
});
aliyunListPlayer.setOnTrackChangedListener(new IPlayer.OnTrackChangedListener() {
    @Override
    public void onChangedSuccess(TrackInfo trackInfo) {
        //切换音视频流或者清晰度成功
    }
    @Override
    public void onChangedFail(TrackInfo trackInfo, ErrorInfo errorInfo) {
        //切换音视频流或者清晰度失败
    }
});
aliyunListPlayer.setOnStateChangedListener(new IPlayer.OnStateChangedListener() {
    @Override
    public void onStateChanged(int newState) {
        //播放器状态改变事件
    }
});
aliyunListPlayer.setOnSnapShotListener(new IPlayer.OnSnapShotListener() {
    @Override
```

```
    @Override
    public void onSnapShot(Bitmap bm, int with, int height) {
        //截图事件
    }
});
```

3. 设置预加载个数。

   合理设置预加载个数，能够有效的提高起播的速度。示例如下：

```
//设置预加载个数。总共加载的个数为： 1 + count*2。
aliyunListPlayer.setPreloadCount(int count);
```

4. 添加或移除多个播放源。

   列表播放支持两种播放源：VidSts播放和UrlSource播放。Url为视频的播放地址。示例如下：

   - Url：播放地址可以是第三方点播地址或阿里云点播服务中的播放地址。阿里云播放地址可以调用获取音视频播放地址接口获取。建议您集成点播服务端SDK来获取音视频播放地址，免去自签名的麻烦。调用接口获取音视频播放地址的示例请参见开发者门户。
   - Vid：音视频ID，可以在音视频上传完成后通过控制台（路径：媒资库 > 音/视频。）或服务端接口（搜索媒体信息）获取。

```
//添加VidSts播放源
aliyunListPlayer.addVid(String videoId, String uid);
//添加UrlSource播放源
aliyunListPlayer.addUrl(String url, String uid);
//移除某个源
aliyunListPlayer.removeSource(String uid);
```

> **说明**
>
> - 不支持VidAuth播放和VidMps播放方式。
> - uid是视频的唯一标志。用于区分视频是否一样。如果uid一样，则认为视频是一样的。如果播放出现串流的情况，请注意查看是不是在不同的界面设置了同一个uid。uid没有格式限制，任意字符串都行。

5. 设置显示View。

   播放器支持SurfaceView和TextureView，任选其中一种即可。

   - 设置SurfaceView，示例如下：

```
SurfaceView surfaceView = findViewById(R.id.surface_view);
surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        aliyunListPlayer.setSurface(holder.getSurface());
    }
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int h
eight) {
        aliyunListPlayer.surfaceChanged();
    }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        aliyunListPlayer.setSurface(null);
    }
});
```

- 设置TextureView，示例如下：

```
TextureView textureView = findViewById(R.id.texture_view);
textureView.setSurfaceTextureListener(new TextureView.SurfaceTextureListener() {
    @Override
    public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int
height) {
        aliyunListPlayer.setSurface(new Surface(surface));
    }
    @Override
    public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width, in
t height) {
        aliyunListPlayer.surfaceChanged();
    }
    @Override
    public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
        aliyunListPlayer.setSurface(null);
        return false;
    }
    @Override
    public void onSurfaceTextureUpdated(SurfaceTexture surface) {
    }
});
```

6. 播放视频源。

   添加了一个或多个播放源之后，调用 `moveTo` 便可以自动播放某个视频源。示例如下：

```
//url时使用此接口
aliyunVodPlayer.moveTo(String uid);
//vid的时候使用此接口，需要传递stsInfo即STS临时凭证和临时AK对，需要提前获取，获取方式请参见创建
角色并进行STS临时授权。
aliyunVodPlayer.moveTo(String uid, StsInfo info);
```

7. 播放上一个、下一个视频。

   播放器提供了 `moveToPrev` ， `moveToNext` 接口用于播放上一个，下一个视频。示例如下：

```
//移动到下一个视频。 注意：只能用于url的源。这个方法不适用于vid的播放。
aliyunVodPlayer.moveToNext();
//移动到上一个视频。注意：只能用于url的源。这个方法不适用于vid的播放。
aliyunVodPlayer.moveToPrev();
//移动到下一个视频。注意：只能用于vid的播放。
aliyunVodPlayer.moveToNext(StsInfo info);
// 移动到上一个视频。注意：只能用于vid的播放。
aliyunVodPlayer.moveToPrev(StsInfo info);
```

外挂字幕

Android播放器SDK支持添加和切换外挂字幕，现已支持SRT、SSA、ASS3种格式。示例如下：

1. 创建显示字幕的View。

   根据不同的字幕格式创建不同的View。

```
//用于显示srt字幕
SubtitleView subtitleView = new SubtitleView(getContext());
//用于显示ASS和SSA字幕
AssSubtitleView assSubtitleView = new AssSubtitleView(getContext());
//将字幕View添加到布局视图中
viewGroup.addView(assSubtitleView);
```

2. 设置字幕相关监听。

```
mAliPlayer.setOnSubtitleDisplayListener(new IPlayer.OnSubtitleDisplayListener() {
    @Override
    public void onSubtitleExtAdded(int trackIndex, String url) { }
    @Override
    public void onSubtitleShow(int trackIndex, long id, String data) {
            // ass 字幕
        assSubtitleView.show(id,data);
        // srt 字幕
        SubtitleView.Subtitle subtitle = new SubtitleView.Subtitle();
        subtitle.id = id + "";
        subtitle.content = data;
        subtitleView.show(subtitle);
    }
    @Override
    public void onSubtitleHide(int trackIndex, long id) {
            // ass 字幕
            assSubtitleView.dismiss(id);
        // srt 字幕
        subtitleView.dismiss(id + "");
    }
    @Override
    public void onSubtitleHeader(int trackIndex, String header) { }
});
```

3. 添加字幕。

```
mAliPlayer.addExtSubtitle(url);
```

4. 切换字幕。

```
mAliPlayer.selectTrack(trackIndex);
```

纯音频播放

通过禁用视频播放，达到纯音频播放的效果。在prepare之前配置PlayerConfig。

```
PlayerConfig config = mAliPlayer.getConfig();
config.mDisableVideo = true;  //设置开启纯音频播放
mAliPlayer.setConfig(config);
```

软硬解切换

> 说明 解码方式需要在播放前切换，播放中切换解码方式将不会生效。

Android播放器SDK提供了H.264、H.265的硬解码能力，同时提供了 `enableHardwareDecoder` 提供开关。默认开，并且在硬解初始化失败时，自动切换为软解，保证视频的正常播放。示例如下：

```
//开启硬解。默认开启
mAliyunVodPlayer.enableHardwareDecoder(true);
```

如果从硬解自动切换为软解，将会通过 `onInfo` 回调，示例如下：

```
mApsaraPlayerActivity.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if (infoBean.getCode() == InfoCode.SwitchToSoftwareVideoDecoder) {
            //切换到软解
        }
    }
});
```

网络自适应切换视频清晰度

Android播放器SDK支持多码率自适应HLS、DASH视频流。在 `prepare` 成功之后，通过 `getMediaInfo` 可以获取到各个码流的信息，即 `TrackInfo` 。示例如下：

```
List<TrackInfo> trackInfos  = aliyunVodPlayer.getMediaInfo().getTrackInfos();
```

在播放过程中，可以通过调用播放器的 `selectTrack` 方法切换播放的码流，取值为AUTO_SELECT_INDEX时，为多码率自适应。示例如下：

```
int index = trackInfo.getIndex();
//多码率切换
aliyunVodPlayer.selectTrack(index);
//多码率切换并自适应
aliyunVodPlayer.selectTrack(TrackInfo.AUTO_SELECT_INDEX);
```

切换的结果会在 `OnTrackChangedListener` 监听之后会回调（在调用 `selectTrack` 之前设置）。示例如下：

```
aliyunVodPlayer.setOnTrackChangedListener(new IPlayer.OnTrackChangedListener() {
    @Override
    public void onChangedSuccess(TrackInfo trackInfo) {
        //切换成功
    }
    @Override
    public void onChangedFail(TrackInfo trackInfo, ErrorInfo errorInfo) {
        //切换失败。失败原因通过errorInfo.getMsg()获取
    }
});
```

截图

Android播放器SDK提供了对当前视频截图的功能，由 `snapshot` 接口实现。截取的是原始的数据，并转为 `bitmap` 返回。回调接口为 `OnSnapShotListener` 。示例如下：

```
//设置截图回调
aliyunVodPlayer.setOnSnapShotListener(new OnSnapShotListener(){
    @Override
    public void onSnapShot(Bitmap bm, int with, int height){
        //获取到的bitmap以及图片的宽高。
    }
});
//截取当前播放的画面
aliyunVodPlayer.snapshot();
```

试看

Android播放器SDK通过配合点播服务配置，可以实现试看功能，支持VidSts和VidAuth（视频点播推荐使用此方式）两种播放方式。如何配置和使用试看功能，请参见点播试看。

配置试看功能之后，通过 `VidPlayerConfigGen.setPreviewTime()` 方法设置播放器的试看时长。以VidSts播放方式为例，示例如下：

```
VidSts vidSts = new VidSts;
....
VidPlayerConfigGen configGen = new VidPlayerConfigGen();
configGen.setPreviewTime(20);//20秒试看
vidSts.setPlayConfig(configGen);//设置给播放源
...
```

当设置试看的时长，通过Android播放器SDK播放视频时，服务端将不会返回完整的视频内容，而是返回试看时间段的内容。

> **说明**
>
> - VidPlayerConfigGen支持设置服务端支持的请求参数。请参见 请求参数说明 。
> - FLV和MP3格式视频暂时不支持试看。

设置黑名单

Android播放器SDK提供了硬解的黑名单机制。对于明确不能使用硬解播放的机器，可以直接使用软解，避免了无效的操作。示例如下：

```
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.model="Lenovo K320t";
AliPlayerFactory.addBlackDevice(BlackType.HW_Decode_H264 ,deviceInfo );
```

> **说明** 退出App之后，黑名单自动失效。

设置Referer

Android播放器SDK支持设置Referer，配合控制台的黑白名单Referer，可以控制访问权限，由 `PlayerConfig` 方法设置请求Referer。播放器SDK的设置示例如下：

```
//先获取配置
PlayerConfig config = mAliyunVodPlayer.getConfig();
//设置referer，示例：http://example.aliyundoc.com。（注意：设置referer时，需要加上前面的协议部分。）
config.mReferrer = referrer;
....//其他设置
  //设置配置给播放器
mAliyunVodPlayer.setConfig(config);
```

设置UserAgent

Android播放器SDK提供了 `PlayerConfig` 用来设置请求UA。设置之后，播放器请求的过程中将会带上UA信息。示例如下：

```
//先获取配置
PlayerConfig config = mAliyunVodPlayer.getConfig();
//设置UA
config.mUserAgent = "需要设置的UserAgent";
....//其他设置
  //设置配置给播放器
mAliyunVodPlayer.setConfig(config);
```

配置网络重试时间和次数

支持设置Android播放器SDK的网络超时的时间和重试次数，由 `PlayerConfig` 方法实现。示例如下：

```
//先获取配置
PlayerConfig config = mAliyunVodPlayer.getConfig();
//设置网络超时时间，单位：毫秒
config.mNetworkTimeout = 5000;
//设置超时重试次数。每次重试间隔为networkTimeout。networkRetryCount=0则表示不重试，重试策略app决定，默认值为2
config.mNetworkRetryCount=2;
....//其他设置
  //设置配置给播放器
mAliyunVodPlayer.setConfig(config);
```

> **说明**
>
> - 如果设置了NetworkRetryCount，若此时发生网络问题，导致出现loading后，那么将会重试NetworkRetryCount次，每次的间隔时间为mNetworkTimeout。
> - 如果重试多次之后，还是loading的状态，那么就会回调 `onError` 事件，此时，ErrorInfo.getCode()=ErrorCode.ERROR_LOADING_TIMEOUT。
> - 如果NetworkRetryCount设置为0，当网络重试超时的时候，播放器就会回调 `onInfo` 事件，事件的InfoBean.getCode()=InfoCode.NetworkRetry。 此时，可以调用播放器的 `reload` 方法进行重新加载网络，或者进行其他的处理。

配置缓存和延迟控制

Android播放器SDK通过 `PlayerConfig` 提供了设置缓存和延迟的控制接口。示例如下：

```
//先获取配置
PlayerConfig config = mAliyunVodPlayer.getConfig();
//最大延迟。注意：直播有效。当延时比较大时，播放器sdk内部会追帧等，保证播放器的延时在这个范围内。
config.mMaxDelayTime = 5000;
// 最大缓冲区时长。单位ms。播放器每次最多加载这么长时间的缓冲数据。
config.mMaxBufferDuration = 50000;
//高缓冲时长。单位ms。当网络不好导致加载数据时，如果加载的缓冲时长到达这个值，结束加载状态。
config.mHighBufferDuration = 3000;
// 起播缓冲区时长。单位ms。这个时间设置越短，起播越快。也可能会导致播放之后很快就会进入加载状态。
config.mStartBufferDuration = 500;
....//其他设置
//往前缓存的最大时长。单位ms。默认为0。
config.mMaxBackwardBufferDurationMs = 0;
//设置配置给播放器
mAliyunVodPlayer.setConfig(config);
```

2022.02.22新增往前缓存的最大时长

> **注意** 三个缓冲区时长的大小关系必须为：mStartBufferDuration ≤ mHighBufferDuration ≤ mMaxBufferDuration。

设置HTTP Header

通过 `PlayerConfig` 方法，可以给播放器中的请求加上HTTP的header参数。示例如下：

```
//先获取配置
PlayerConfig config = mAliyunVodPlayer.getConfig();
//定义header
String[] headers = new String[1];
headers[0]="Host:example.com";//比如需要设置Host到header中。
//设置header
config.setCustomHeaders(headers);
....//其他设置
  //设置配置给播放器
mAliyunVodPlayer.setConfig(config);
```

性能
本地缓存

Android播放器SDK提供了本地缓存（边播边缓存）的功能，能够让用户重复播放视频时，达到省流量的目的。只需在 `prepare` 之前给播放器配置 `CacheConfig` 即可实现此功能。示例如下：

```
CacheConfig cacheConfig = new CacheConfig();
//开启缓存功能
cacheConfig.mEnable = true;
 //能够缓存的单个文件最大时长。超过此长度则不缓存
cacheConfig.mMaxDurationS =100;
//缓存目录的位置
cacheConfig.mDir = "缓存的文件目录";
 //缓存目录的最大大小。超过此大小，将会删除最旧的缓存文件
cacheConfig.mMaxSizeMB = 200;
//设置缓存配置给到播放器
mAliyunVodPlayer.setCacheConfig(cacheConfig);
```

缓存成功之后，以下情况将会利用缓存文件（必须已经设置了 `setCacheConfig` ）

- 如果设置了循环播放，即 `setLoop(true)` ，那么第二次播放的时候，将会自动播放缓存的文件。
- 缓存成功后，重新创建播放器，播放同样的资源，也会自动使用缓存文件。

> **说明** vid的缓存文件是通过vid等信息定位的，所以对于vid的缓存文件，将需要网络请求信息之后才能确定使用哪个缓存文件。

播放器提供了获取缓存文件路径的接口：

API接口	描述	参数	返回值
`public String getCacheFilePath(String URL)`	根据URL获取缓存的文件名。必须先调用 `setCacheConfig` 接口才能获取到。	URL	最终缓存的文件绝对路径。
`public String getCacheFilePath(String vid, String format, String definition, int previewTime)`	根据vid获取缓存的文件名。必须先调用 `setCacheConfig` 接口才能获取到。	• vid：视频ID • format：视频格式 • definition：视频清晰度 • previewTime：试看时长	最终缓存的文件绝对路径。

边播边缓存的使用有限制条件，具体如下：

- 对于UrlSource播放方式。如果是HLS（即M3U8）地址，将不会缓存。如果是其他支持的格式，则根据缓存配置进行缓存。
- 对于VidAuth、VidSts播放方式，将会根据缓存配置进行缓存。
- 播放器读取完全部的数据则视为缓存成功。如果在此之前，调用 `stop` ，或者出错 `onError` ，则缓存将会失败。
- cache内的seek的操作不会影响缓存结果。cache外的seek会导致缓存失败。
- 如果视频源是加密的，此时如果加密文件与App信息不一致，将会缓存失败。
- cache的结果回调，会通过 `onInfo` 回调。

```
mAliPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if(infoBean.getCode() == InfoCode.CacheSuccess){
            //缓存成功事件。
        }else if(infoBean.getCode() == InfoCode.CacheError){
            //缓存失败事件
        }
    }
});
```

预加载

Android播放器SDK提供预加载功能，是对本地缓存（边播边缓存）功能的升级，通过设置视频缓存的内存占用大小，更能提升视频的起播速度。

预加载功能的使用说明如下所示：

- 目前支持MP4、MP3、FLV、HLS（单码率视频流）等单个媒体文件的加载。
- 仅支持UrlSource播放方式播放视频的预加载，暂不支持VidAuth、VidSts方式播放视频的预加载。

1. 开启本地缓存功能。

   本地缓存功能默认关闭，如需使用，需要手动开启。通过AliPlayerGlobalSettings中的 `enableLocalCache` 控制。

```
/**
 *  开启本地缓存，开启之后，会缓存到本地文件中。
 *
 *  @param enable - 本地缓存功能开关。true：开启本地缓存，false：关闭，默认关闭。
 *  @param maxBufferMemoryKB - 设置能够缓存的单个视频源文件的最大内存占用大小，超过此大小则不缓存，单位：KB。
 *  @param localCacheDir - 本地缓存的文件目录，为绝对路径。
 */
public static void enableLocalCache(boolean enable,
                                     int maxBufferMemoryKB,
                                     java.lang.String localCacheDir)
 /**
  * 本地缓存文件清理相关配置。
  *
  * @param expireMin - 设置缓存的过期时间。单位：分钟，默认值30天，过期的缓存不管容量多大，都会被删除。
  * @param maxCapacityMB - 最大缓存容量。单位：兆，默认值20GB，在清理时，如果缓存总容量超过此大小，则会以cacheItem为粒度，按缓存的最后时间排序，一个一个的删除最旧的缓存文件，直到小于等于最大缓存容量。
  * @param freeStorageMB - 磁盘最小空余容量。单位：兆，默认值0，在清理时，同最大缓存容量，如果当前磁盘容量小于该值，也会按规则一个一个的删除缓存文件，直到freeStorage大于等于该值或者所有缓存都被清理掉。
  */
 public static void setCacheFileClearConfig(long expireMin,
                                             long maxCapacityMB,
                                             long freeStorageMB)
  /**
   * 设置加载url的hash值回调。如果不设置，SDK使用md5算法。
   */
  public static void
setCacheUrlHashCallback(AliPlayerGlobalSettings.OnGetUrlHashCallback cb)
```

> **说明**
>
> - 如果视频播放URL带有鉴权参数，预加载和播放时鉴权参数会变化，可以将URL的鉴权参数去掉后再计算md5值。例如：带有鉴权参数的视频播放URL为 `http://****.mp4?aaa` ，则加载时使用 `http://****.mp4` 计算md5值。
> - 如果服务器同时支持HTTP和HTTPS协议，但是不同的协议指向的媒体文件是同一个，则可以将请求头去掉或者统一后再计算md5值。例如：
>   - 视频播放URL为 `https://****.mp4` 和 `http://****.mp4` ，则加载时使用 `***.mp4` 计算md5值。
>   - 视频播放URL为 `https://****.mp4` ，加载时统一为 `http://****.mp4` 后再计算md5值。

2. **获取AliMediaLoader实例。**

   AliMediaLoader实例为单例，即无论获取多少次，创建的都是同一个实例。

   ```
   MediaLoader mediaLoader = MediaLoader.getInstance();
   ```

3. **配置AliMediaLoader。**

   配置回调，并开始加载。

```
/**
 * 设置加载状态回调
 */
mediaLoader.setOnLoadStatusListener(new MediaLoader.OnLoadStatusListener() {
    @Override
    public void onError(String url, int code, String msg) {
        //加载出错
    }
    @Override
    public void onCompleted(String s) {
        //加载完成
    }
    @Override
    public void onCanceled(String s) {
        //加载取消
    }
});
/**
 * 开始加载文件。异步加载。可以同时加载多个视频文件。
 * @param url - 视频文件地址。
 * @param duration - 加载的时长大小，单位：毫秒。
 */
mediaLoader.load("url","duration");
```

4. （可选）删除加载文件。

   可按需删除加载文件，以节省空间。Android播放器SDK不提供删除接口，需要在App删除加载目录下的文件。

获取下载速度

指获取当前播放视频的下载速度，在onInfo回调中获取，由 `getExtraValue` 接口实现。示例如下：

```
mAliPlayer.setOnInfoListener(new IPlayer.OnInfoListener() {
    @Override
    public void onInfo(InfoBean infoBean) {
        if(infoBean.getCode() == InfoCode.CurrentDownloadSpeed){
            //当前下载速度
            long extraValue = infoBean.getExtraValue();
        }
    }
});
```

网络特性
用HTTP/2

Android播放器SDK支持使用HTTP/2协议，该协议通过多路复用，避免队头阻塞，以改善播放性能。示例如下：

```
AliPlayerGlobalSettings.setUseHttp2(true);
```

视频下载

Android播放器SDK提供了点播服务视频的下载功能，能够通过VidSts和VidAuth方式下载点播服务上的视频。同时，下载的方式提供了安全下载和普通下载的模式（可登录点播控制台，选择配置管理 > 分发加速配置 > 下载设置配置）。

- 普通下载
  下载后的视频数据未经过阿里云加密，用户可以用第三方播放器播放。
- 安全下载
  下载后的视频数据经过阿里云加密。第三方播放器无法播放。仅支持使用阿里云的播放器进行播放。

1. 创建并设置下载器创建下载器，通过AliDownloaderFactory创建。示例如下：

```
AliMediaDownloader mAliDownloader = null;
......
//创建下载器
mAliDownloader = AliDownloaderFactory.create(getApplicationContext());
//配置下载保存的路径
mAliDownloader.setSaveDir("保存的文件夹地址");
```

下载SDK支持私有加密的下载。为了保证安全性，需要配置一个加密校验文件到SDK（建议在Application中配置一次即可）。示例如下：

```
PrivateService.initService(getApplicationContext(),  "encryptedApp.dat所在的文件路径");
```

> 注意 如果是安全下载，并且加密校验文件与App信息不一致，会导致下载失败。

2. 设置监听事件。

下载器提供了多个事件监听。示例如下：

```
mAliDownloader.setOnPreparedListener(new AliMediaDownloader.OnPreparedListener() {
    @Override
    public void onPrepared(MediaInfo mediaInfo) {
        //准备下载项成功
    }
});
mAliDownloader.setOnProgressListener(new AliMediaDownloader.OnProgressListener() {
    @Override
    public void onDownloadingProgress(int percent) {
        //下载进度百分比
    }
    @Override
    public void onProcessingProgress(int percent) {
        //处理进度百分比
    }
});
mAliDownloader.setOnErrorListener(new AliMediaDownloader.OnErrorListener() {
    @Override
    public void onError(ErrorInfo errorInfo) {
        //下载出错
    }
});
mAliDownloader.setOnCompletionListener(new
AliMediaDownloader.OnCompletionListener() {
    @Override
    public void onCompletion() {
        //下载成功
    }
});
```

3. 准备下载源。

   通过 `preapre` 方法准备下载源。下载源支持VidSts和VidAuth两种方式。以VidSts举例，示例如下：

```
//创建VidSts
VidSts aliyunVidSts = new VidSts();
aliyunVidSts.setVid(视频vid);
aliyunVidSts.setAccessKeyId(临时akId);
aliyunVidSts.setAccessKeySecret(临时akSecret);
aliyunVidSts.setSecurityToken(安全token);
aliyunVidSts.setRegion(接入区域);
//准备下载源
mAliDownloader.prepare(aliyunVidSts)
```

4. 准备成功后选择下载项。

   准备成功后，会回调 `OnPreparedListener` 方法。选择某个Track进行下载，示例如下：

```
public void onPrepared(MediaInfo mediaInfo) {
    //准备下载项成功
    List<TrackInfo> trackInfos = mediaInfo.getTrackInfos();
    //比如：下载第一个TrackInfo
    mAliDownloader.selectItem(trackInfos.get(0).getIndex());
}
```

5. 更新下载源并开始下载。

   为了防止VidSts和VidAuth过期，建议更新下载源的信息后开始下载。示例如下：

```
//更新下载源
mAliDownloader.updateSource(VidSts);
//开始下载
mAliDownloader.start();
```

6. 下载成功或失败后，释放下载器。

   下载成功后，调用 `release` 释放下载器。在 `onCompletion` 或者 `onError` 回调中。示例如下：

```
mAliDownloader.stop();
mAliDownloader.release();
```

7. 删除下载的文件。

   下载过程中，或者下载完成后，可以删除下载的文件。示例如下：

```
//通过对象删除文件
mAliDownloader.deleteFile();
//通过静态方法删除
AliDownloaderFactory.deleteFile("下载文件夹路径",vid, format,index);
```

视频加密播放

点播视频支持HLS标准加密、阿里云私有加密和DRM加密，直播视频仅支持DRM加密。加密播放请参见 视频加密播放。

Native RTS播放

Android播放器SDK集成Native RTS SDK实现Native端低延时直播功能，详情请参见 阿里云播放器SDK集成Native RTS SDK实现说明（Android端）。

异常处理

使用阿里云播放器播放视频发生异常时，可借助单点探查功能快速定位问题，详细内容，请参见 单点探查。2022.01.28：新增单点探查功能，补充相关说明。

相关文档
接口说明

5.5. SDK升级

把下面两个升级的放到这里的子目录下。

5.5.1. 升级至

本文介绍了Android播放器SDK从低于4.5.0版本升级至4.5.0版本的步骤及SDK在4.5.0发生的重要变更。

背景信息
Android播放器SDK在4.5.0版本对接口、方法和类进行了大幅改动。如果您正在使用 4.5.0以下版本，建议通过本文提供的方法升级至4.5.0。详细变化请参见下文内容：

- 升级步骤-本地依赖
- 升级步骤-gradle依赖
- 播放器变化
- 依赖包变化
- 混淆配置变化
- 类名与方法变化
- 监听事件变化
- 主要API变化

升级路径
本文以3.4.10版本为例介绍升级至4.5.0版本的操作步骤。Andorid播放器SDK 4.5.0版本及后续的版本SDK仅类名和方法有变化，版本号有区别。4.5.0升级至5.1.4版本接口变化详情请参见 升级Andorid播放器至5.1.0 。低于4.5.0版本直接升级至5.1.4版本暂无参考文档，如有需要请 提交工单 联系阿里云技术支持。

升级步骤-本地依赖
如果您采用了本地依赖方式集成Android播放器SDK，升级至4.5.0版本需要替换本地依赖包及gradle依赖包。替换完成后按指引修改代码完成升级。

1. 替换本地依赖包及gradle依赖包。

i. **替换本地依赖包。**

将 *AlivcPlayer-3.4.10.aar*、 *AlivcReporter-1.2.aar*、 *AliyunVodPlayer-3.4.10.aar***替换为** *AliyunPlayer-4.5.0-full.aar***。**

替换完成：

ii. **增加阿里云的**Maven**地址。**

在项目的*build.gradle***文件中，在** `buildscript` **和** `allprojects` **两处增加阿里**Maven**地址，示例如下：**

```
maven { url "http://maven.aliyun.com/nexus/content/repositories/releases" }
```

iii. **替换**gradle**依赖包。**

将OSS**的依赖删除，增加**conan**的依赖。修改后如下：**

2. **集成短视频。**

如果集成Android**播放器**SDK**的同时也集成短视频**SDK**，那么播放器的**SDK**需要依赖两个包：** *AliyunPlayer-4.5.0-part***和***com.aliyun.video.android:AlivcFFmpeg:1.0.0***（这个包是为了使用与短视频**SDK**共通的**ffmpeg**版本）。**

> **说明** 如果集成时使用了错误的SDK包，会导致ffmpeg冲突。

3. **修改代码。**

依赖包替换完成之后，编译工程就会出错。这个时候，需要使用播放器4.5.0**版本**SDK**中的类和方法。此处变化比较大，主要是包名的变化与**API**的变化。下面是几个需要重点修改的地方：**

i. **更改创建播放器方式**

播放器4.5.0版本SDK创建播放器对象类型改为 `AliPlayer`，通过 `AliPlayerFactory` 类进行创建。示例如下：

```
//定义一个播放器对象
AliPlayer mAliyunVodPlayer;
...
//创建播放器对象
mAliyunVodPlayer =
AliPlayerFactory.createAliPlayer(getContext().getApplicationContext());
```

ii. **修改监听事件**

播放器3.4.10版本SDK，监听事件都是位于 `IAliyunVodPlayer` 类下面。播放器4.5.0版本SDK，所有的监听事件都是在 `IPlayer` 类下面。类名基本保持一致，有些有变化。所以替换监听事件需要替换类名。详细变化情况请参见监听事件变化。此处以 `OnPreparedListener` 为例：

```
//设置准备回调
mAliyunVodPlayer.setOnPreparedListener(new IAliyunVodPlayer.OnPreparedListener() {
    @Override
    public void onPrepared() {
        if (mAliyunVodPlayer == null) {
```

3.4.10的类名

修改类名后：

```
mAliyunVodPlayer = AliPlayerFactory.createAliPlayer(getContext().getApplicationCo
//设置准备回调
mAliyunVodPlayer.setOnPreparedListener(new IPlayer.OnPreparedListener() {
    @Override
    public void onPrepared() {
        if (mAliyunVodPlayer == null) {
            return;
```

4.5.0版本类名

4. **设置播放源。**

播放器3.4.10版本SDK通过 `prepareAsync` 方法直接传值。播放器4.5.0版本SDK改为 `AliPlayer.setDataSource` 方法设置。设置完之后，调用 `AliPlayer.prepare` 方法准备播放源。示例如下：

```
/**
 * 设置UrlSource数据源
 */
public void setDataSource(UrlSource urlSource);
/**
 * 设置vidSts数据源
 */
public void setDataSource(VidSts vidSts);
/**
 * 设置vidAuth数据源
 */
public void setDataSource(VidAuth vidAuth);
/**
 * 设置mps数据源
 */
public void setDataSource(VidMps vidMps);
```

设置直播时移：通过 `AliLiveShiftPlayer.setDataSource` 方法设置时移源。示例如下：

```
/**
 * 设置数据源
 */
public void setDataSource(LiveShift liveShift);
```

> **说明** 如果由AlivcMediaPlayer升级到新版本播放，改用UrlSource播放即可。

升级步骤-gradle依赖

如果您采用了gradle依赖方式集成Android播放器SDK，升级至4.5.0版本需要删除本地依赖并替换gradle依赖包。

1. 删除本地依赖并替换gradle依赖包。
   i. 删除本地依赖。

   删除 *AlivcPlayer-3.4.10.aar*、 *AlivcReporter-1.2.aar*、 *AliyunVodPlayer-3.4.10.aar*依赖包（删除框中的内容）。

   ii. 增加阿里云的Maven地址。

   在项目的 *build.grade*文件中，在 `buildscript` 和 `allprojects` 两处增加阿里云的Maven地址： 缺陷修复，maven=>Maven。无需送翻。

   ```
   maven { url "http://maven.aliyun.com/nexus/content/repositories/releases" }
   ```

   iii. 替换gradle依赖包。

   修改App的*build.gradle*文件， `dependencies` 节点中增加对aar的引用。示例如下：

   ```
   implementation 'com.aliyun.sdk.android:AliyunPlayer:4.5.0-full' implementation
   'com.alivc.conan:AlivcConan:0.9.5'
   ```

2. 集成短视频。

   如果集成Android播放器SDK的同时也集成短视频SDK，那么播放器的SDK需要依赖两个包： *AliyunPlayer-4.5.0-part*和*com.aliyun.video.android:AlivcFFmpeg:1.0.0*（这个包是为了使用与短视频SDK共通的ffmpeg版本）。

   > **说明** 如果集成时使用了错误的SDK包，会导致ffmpeg冲突。

3. 修改代码。

   依赖包替换完成之后，编译工程就会出错。这个时候，需要使用播放器4.5.0版本SDK中的类和方法。此处变化比较大，主要是包名的变化与API的变化。下面是几个需要重点修改的地方：

i. **更改创建播放器方式**

播放器4.5.0版本SDK创建播放器对象类型改为 `AliPlayer` ，通过 `AliPlayerFactory` 类进行创建。示例如下：

```
//定义一个播放器对象
AliPlayer mAliyunVodPlayer;
...
//创建播放器对象
mAliyunVodPlayer =
AliPlayerFactory.createAliPlayer(getContext().getApplicationContext());
```

ii. **修改监听事件**

播放器3.4.10版本SDK，监听事件都是位于 `IAliyunVodPlayer` 类下面。播放器4.5.0版本SDK，所有的监听事件都是在 `IPlayer` 类下面。类名基本保持一致，有些有变化。所以替换监听事件需要替换类名。详细变化情况请参见监听事件变化。此处以 `OnPreparedListener` 为例：

```
//设置准备回调
mAliyunVodPlayer.setOnPreparedListener(new IAliyunVodPlayer.OnPreparedListener() {
    @Override
    public void onPrepared() {
        if (mAliyunVodPlayer == null) {
```

3.4.10的类名

修改类名后：

```
mAliyunVodPlayer = AliPlayerFactory.createAliPlayer(getContext().getApplicationCo
//设置准备回调
mAliyunVodPlayer.setOnPreparedListener(new IPlayer.OnPreparedListener() {
    @Override
    public void onPrepared() {
        if (mAliyunVodPlayer == null) {
            return;
```

4.5.0版本类名

4. **设置播放源。**

播放器3.4.10版本SDK通过 `prepareAsync` 方法直接传值。播放器4.5.0版本SDK改为 `AliPlayer.setDataSource` 方法设置。设置完之后，调用 `AliPlayer.prepare` 方法准备播放源。示例如下：

```
/**
 * 设置UrlSource数据源
 */
public void setDataSource(UrlSource urlSource);
/**
 * 设置vidSts数据源
 */
public void setDataSource(VidSts vidSts);
/**
 * 设置vidAuth数据源
 */
public void setDataSource(VidAuth vidAuth);
/**
 * 设置mps数据源
 */
public void setDataSource(VidMps vidMps);
```

设置直播时移：通过 `AliLiveShiftPlayer.setDataSource` 方法设置时移源。示例如下：

```
/**
 * 设置数据源
 */
public void setDataSource(LiveShift liveShift);
```

说明 如果由AlivcMediaPlayer升级到新版本播放，改用UrlSource播放即可。

播放器变化

变化	低于4.5.0	4.5.0
统一播放器	提供了两种播放器，分别用于URL播放与VID视频的播放。 • AlivcMediaPlayer播放器 • AliyunVodPlayer播放器	在SDK4.5.0中，不再区分播放器，统一为AliPlayer端播放器对URL视频和VID视频进行播放。

依赖包变化

变化	低于4.5.0	4.5.0
依赖包	3个本地依赖和1个gradle依赖： • AlivcPlayer-3.x.x.aar文件 • AlivcReporter-1.2.aar文件 • AliyunVodPlayer-3.x.x.aar文件 • gradle依赖：compile 'com.aliyun.dpa:oss-android-sdk:+'	• AliyunPlayer-4.5.0-full.aar文件 • implementation 'com.alivc.conan:AlivcConan:0.9.5'

混淆配置变化

Android播放器SDK4.5.0版本混淆配置示例如下：

```
 -keep class com.alivc.**{*;} -keep class com.aliyun.**{*;} -keep class com.cicada.**{*;} -dontwarn com.alivc.** -dontwarn com.aliyun.**
```

类名与方法变化

Android播放器SDK4.5.0版本，修改了部分类名，增加了新的功能和事件回调。同时，也移除了低于SDK4.5.0版本中的部分接口。

监听事件变化

大部分监听事件替换类名即可。对于有删减或者不同的，需要修改对应的接口实现。下面是两个版本监听事件的对比：

事件名	功能	变化情况	示例代码
OnPreparedListener	准备成功事件	无变化	不涉及
OnSeekCompleteListener	拖动结束事件	无变化	不涉及
OnCompletionListener	播放完成事件	无变化	不涉及

事件名	功能	变化情况	示例代码
OnFirstFrameStartListener	首帧显示事件	OnRenderingStartListener	`public interface OnRenderingStartListener { void onRenderingStart(); } public void setOnRenderingStartListener(OnRenderingStartListener l);`
			`public interface OnStateChangedListener { void onStateChanged(int newState); } public void setOnStateChangedListener(OnStateChangedListener l);` `OnStateChangedListener` 接口回调中的 `newState` 值，表示播放器的新的状态，包含了多种状态值。状态的值定义在IPlayer中。示例如下：

事件名	功能	变化情况	示例代码
OnStoppedListener	播放停止事件	OnStateChangedListener	未知状态 */ public static final int unknow = -1; /** * 空状态。刚创建出来的状态。 */ public static final int idle = 0; /** * 初始化了的状态，设置播放源之后的状态 */ public static final int initalized = 1;// /** * 准备成功的状态 */ public static final int prepared = 2; /** * 正在播放的状态 */ public static final int started = 3; /** * 播放已暂停的状态 */ public static final int paused = 4; /** * 播放已停止的状态 */ public static final int stopped = 5; /** * 播放完成状态 */ public static final int completion = 6; /** * 出错状态 */ public static final int error = 7;
OnPcmDataListener	PCM音频数据回调事件	事件被删除	不涉及

事件名	功能	变化情况	示例代码
OnCircleStartListener	**循环播放开始事件**	OnInfoListener	`public interface OnInfoListener { void onInfo(InfoBean infoBean); } public void setOnInfoListener(OnInfoListener l);` **说明** 回调中的InfoBean.getCode() = InfoCode.LoopingStart被回调时，表示循环播放事件。

事件名	功能	变化情况	示例代码
OnLoadingListener	**视频加载事件**	OnLoadingStatusListener	`public interface OnLoadingStatusListener { void onLoadingBegin(); void onLoadingProgress(int percent, float netSpeed); //百分比，kbps void onLoadingEnd(); } public void setOnLoadingStatusListener(OnLoadingStatusListener l);` **说明** onLoadingProgress中的netSpeed目前值为0。

事件名	功能	变化情况	示例代码
OnBufferingUpdateListener	视频缓冲进度事件	OnInfoListener	`public interface OnInfoListener { void onInfo(InfoBean infoBean); } public void setOnInfoListener(OnInfoListener l);` **说明** 回调中的InfoBean.getCode() = InfoCode.BufferedPosition被回调时，表示缓冲进度事件。通过 `InfoBean.getExtraValue()` 获取缓冲进度的值。同时，如果 `InfoBean.getCode() = InfoCode.CurrentPosition` 被回调时，表示播放进度事件。通过 `InfoBean.getExtraValue()` 获取播放进度的值。
OnErrorListener	播放错误事件	OnErrorListener	`public interface OnErrorListener { void onError(ErrorInfo errorInfo); } public void setOnErrorListener(OnErrorListener l);` **说明** ErrorInfo为错误信息，可以通过 `ErrorInfo.getCode()` 和 `ErrorInfo.getMsg()` 获取错误码与错误信息。

事件名	功能	变化情况	示例代码
OnInfoListener	播放信息事件	OnInfoListener	public interface OnInfoListener { void onInfo(InfoBean infoBean); } public void setOnInfoListener (OnInfoListener l); InfoBean为信息类，可以通过 InfoBean.getCode() 和 InfoBean.getExtraMsg() ，获取具体的信息类型等。InfoCode包含： /** * 未知。 */ Unknown(-1); /** * 循环播放开始。无额外信息。 */ LoopingStart(0), /** * 缓冲位置。额外值为当前缓冲位置。单位：毫秒。 */ BufferedPosition(1), /** * 当前播放位置。额外值为当前播放位置。单位：毫秒。 */ CurrentPosition(2), /** * 开始自动播放。无额外信息。 */ AutoPlayStart(3), /** * 设置了硬解，但是切换为软解。额外信息为描述信息。 */ SwitchToSoftwareVideoDecoder(100), /** * 音频解码格式不支持。额外信息为描述信息。 */ AudioCodecNotSupport(101), /** * 音频解码器设备失败。额外信息为描述信息。 */ AudioDecoderDeviceError(102), /** * 视频解码格式不支持。额外信息为描述信息。 */

事件名	功能	变化情况	示例代码
			信息。 */ VideoCodecNotSupport(103), /** * 视频解码器设备失败。额外信息为描述信息。 */ VideoDecoderDeviceError(104), /** * 网络失败，需要重试。无额外信息。 */ NetworkRetry(107), /** * 缓存成功。无额外信息。 */ CacheSuccess(108), /** * 缓存失败。额外信息为描述信息。 */ CacheError(109), /** * 系统无可用内存来存放媒体数据 */ LowMemory(110),
OnVideoSizeChangedListener	视频画面变化回调	无变化	不涉及

事件名	功能	变化情况	示例代码
OnChangeQualityListener	**切换清晰度变化**	OnTrackChangedListener	`public interface OnTrackChangedListener { void onChangedSuccess(TrackInfo trackInfo); void onChangedFail(TrackInfo trackInfo, ErrorInfo errorInfo); } public void setOnTrackChangedListener(OnTrackChangedListener l);` **说明** 切换清晰度调用 `selectTrack(index)` 之后，将会回调此方法。
LockPortraitListener	**固定竖屏**	**事件被删除**	**不涉及**
OnRePlayListener	**重播开始事件**	**事件被删除**	**不涉及**

事件名	功能	变化情况	示例代码
OnAutoPlayListener	自动播放开始事件	OnInfoListener	`public interface OnTrackChangedListener { void onChangedSuccess(TrackInfo trackInfo); void onChangedFail(TrackInfo trackInfo, ErrorInfo errorInfo); } public void setOnTrackChangedListener(OnTrackChangedListener l);` 说明 ErrorInfo为错误信息，可以通过 `ErrorInfo.getCode()` 和 `ErrorInfo.getMsg()` 获取错误码与错误信息。过期错误码的值，请参见 错误码查询。
OnTimeShiftUpdaterListener	直播时移变化事件	移动到AliLiveshiftPlayer中	不涉及
OnSeekLiveCompletionListener	直播时移拖动结束事件	移动到AliLiveshiftPlayer中	不涉及
OnTimeExpiredErrorListener	请求信息过期事件	OnErrorListener	`public interface OnErrorListener { void onError(ErrorInfo info); } void setOnErrorListener(IPlayer.OnErrorListener l)`
OnUrlTimeExpiredListener	URL即将过期事件	事件被删除	不涉及
OnTrackReadyListener	流获取成功事件	新增	不涉及
OnSubtitleDisplayListener	字幕显示时间	新增	不涉及

事件名	功能	变化情况	示例代码
OnSnapShotListener	截图结果事件	新增	不涉及

主要API变化

以Android播放器SDK3.4.10版本升级到播放器SDK 4.5.0版本为例。

> 说明 低于SDK 4.5.0版本升级到SDK 4.5.0版本的API变化基本一致。

原方法	新方法	说明
surfaceChanged	redraw	在surface变化的时候调用
getPlayerState	OnStateChangedListener	改为回调获取播放器状态

> 说明 其他API变化请参见 Android播放器SDK 4.5.0。

低于4.5.0版本相关文档

替换链接

如您需要查看低于4.5.0版本接口的相关内容，请参见 Android播放器SDK 。找到您所需版本的更新记录，单击历史版本中的示例代码下载链接即可获取对应信息。

5.5.2. 升级至

本文介绍Android播放器SDK从4.5.0版本升级到5.1.4及以上版本的升级步骤及SDK在5.1.4及以上版本的重要变更。

背景信息

Android播放器SDK4.5.0版本SDK及后续的版本SDK仅类名和方法有变化，版本号有区别。

升级路径

如果您现在使用的是Android播放器SDK 4.5.0版本，想要升级到5.1.4及以上版本时，直接更新SDK依赖包中的版本号即可。具体操作步骤请参见升级步骤。低于4.5.0版本直接升级至5.1.4版本暂无参考文档，如有需要请提交工单联系阿里云技术支持。

升级步骤

如果您现在使用的是4.5.0版本，将 *build.gradle* 中的播放器SDK的两个依赖包 `com.aliyun.sdk.android:AliyunPlayer:4.5.0-full` 和 `com.alivc.conan:AlivcConan:4.5.0` 修改为目标版本对应的版本号即可完成升级。更多SDK信息请参见播放器SDK发布历史 。

> 说明 Android播放器SDK5.3.0及其以后的版本不需要引入 `AlivcConan` 依赖（保持已有版本号不变），只引入 `AliyunPlayer` 依赖（更新至对应的版本号）即可。

类名与方法变化

4.5.0版本的播放器日志相关的接口在 5.1.4及以上中移植到 `com.cicada.player.utils.Logger` 类中，5.1.4及以上中先通过 `Logger.getInstance(context)` 获取 Logger对象，然后调用其他日志相关的方法即可。具体变化情况参见下表：

> 说明 4.5.0的minSDKVersion支持到16， 5.1.4及以上的minSDKVersion支持到18。

4.5.0	5.1.4及以上	描述
VcPlayerLog	相关代码全部删除	日志相关的部分内容。 说明 在5.1.4及以上版本中可通过 getInstance(Context context) 、enableConsoleLog(boolean bEnabled) 、setLogCallback(Logger.OnLogCallback callback) 、setLogLevel(Logger.LogLevel logLevel) 等方法使用日志。
不涉及	新增：getInstance(Context context)	获取Logger单例对象。
enableLog(boolean enable)	enableConsoleLog(boolean bEnabled)	开启播放器日志。
setLogCallback(LogLevel logLevel, IPlayer.OnLogCallback callback)	setLogCallback(Logger.OnLogCallback callback)	设置日志回调。
com.aliyun.player.LogLevel	setLogLevel(Logger.LogLevel logLevel)	设置日志级别。

5.6. 接口说明

本文提供不同版本Android播放器SDK的接口文档链接。

SDK版本	链接
V5.4.5.0	Android播放器SDK接口文档（V5.4.5.0）
V5.4.4.0	Android播放器SDK接口文档（V5.4.4.0）
V5.4.2.0	Android播放器SDK接口文档（V5.4.2.0）
V5.4.1	Android播放器SDK接口文档（V5.4.1）
V5.4.0	Android播放器SDK接口文档（V5.4.0）
V5.3.2	Android播放器SDK接口文档（V5.3.2）
V5.3.0	Android播放器SDK接口文档（V5.3.0）
V5.2.2	Android播放器SDK接口文档（V5.2.2）

SDK版本	链接
V5.2.1	Android播放器SDK接口文档（V5.2.1）
V5.1.6	Android播放器SDK接口文档（V5.1.6）
V5.1.5	Android播放器SDK接口文档（V5.1.5）

6.iOS播放器

6.1. 快速集成

本文提供快速集成iOS播放器SDK的指引。

环境要求

类别	说明
系统版本	支持iOS 8.0及以上版本。
开发工具	建议使用Xcode，本文操作步骤基于Xcode开发。下载地址： Xcode。

前提条件

本地集成SDK时，需要先下载iOS播放器SDK包（包含了iOS播放器SDK及Demo源码），推荐下载使用最新版本。下载地址请参见SDK简介与下载。

解压后的目录结构如下：

文件名	作用
demo	iOS播放器SDK的Demo源码。
doc	iOS播放器SDK接口文档。

sdk	iOS播放器SDK的framework库，只提供了包含bitcode和模拟器的ARM_SIMULATOR文件夹，如果需要不包含bitcode或模拟器的包，请参考以下命令另行下载。
	`cd $(SDK_PATH)  //SDK_PATH：SDK所在路径`
	`sh createMoreKindsOfArch.sh`

ReleaseNote	版本说明。

sdk文件夹下各文件说明如下：

文件名	说明
ARM	带bitcode，不包括模拟器。
ARM_NO_BITCODE	不带bitcode，不包括模拟器。
ARM_SIMULATOR	带bitcode，带模拟器。
ARM_SIMULATOR_NO_BITCODE	不带bitcode，包含i386 x86_64 armv7 arm64架构。

> **注意** 模拟器用于代码调试，发布时，不能使用带模拟器的版本，否则会提交AppStore失败。

Framework说明如下：

Framework	说明
alivcffmpeg.framework	播放器底层，必须。
AliyunMediaDownloader.framework	用于离线下载，非必须。
AliyunPlayer.framework	播放器，必须。
artcSource.framework	支持artc协议，非必须。
artpSource.framework	支持artp协议，非必须。在5.4.5.0版本已经移除了对它的依赖。
RtsSDK.framework	低延迟直播，非必须。
AlivcConan	在5.3.0版本已经移除了对它的依赖。

说明

- 在进行打包时，*dSYM*文件用于crash符号表解析。
- 使用时，alivcffmpeg和AliyunPlayer都是必须的，缺一不可。播放器头文件位于AliyunPlayer。

SDK集成（cocoapods集成）（推荐）

说明 如需同时集成播放器SDK和短视频SDK，将下面代码中的AliPlayerSDK_iOS替换成AliPlayerPartSDK_iOS。AliPlayerPartSDK_iOS不包含ffmpeg，避免了与短视频SDK中的ffmpeg的冲突。

1. 采用pod语句集成播放器SDK，示例代码如下：

```
ruby
  platform:ios, '8.0'
  target 'yourProject' do
    pod 'AliPlayerSDK_iOS'
   end
```

2. （可选）如果需要支持artc协议或播放低延迟直播（RTS）流，请添加以下pod依赖，示例代码如下：

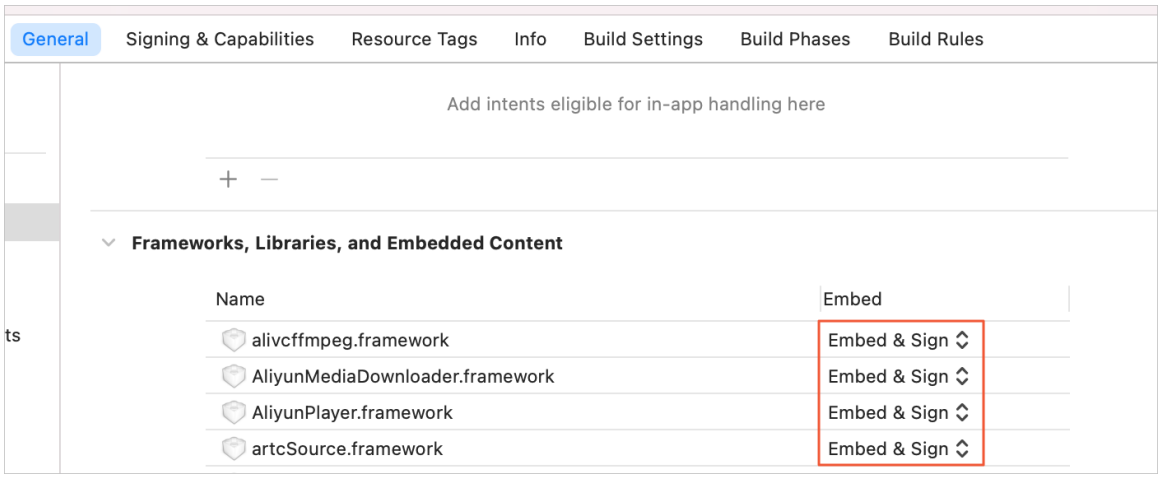
   说明 请确保引入的播放器SDK版本号填写正确，否则将报错并引入失败。

```
ruby
  platform:ios, '8.0'
  target 'yourProject' do
    pod 'AliPlayerSDK_iOS', '5.4.5.0'
    pod 'AliPlayerSDK_iOS_ARTC', '5.4.5.0'
    pod 'RtsSDK', '2.2.0' //此处版本仅供参考，获取最新RTS SDK的版本请参见低延时直播SDK下载。
   end
```

SDK集成（本地集成）

说明 如果您的代码或引用的第三方代码，与alivcffmpeg或AlivcConan的symbol有冲突，可以将alivcffmpeg或AlivcConan从Linked Frameworks and Libraries里删除，app link时可以不依赖这两个framework。AlivcConan从5.3.0版本开始不再需要依赖。

1. 在Xcode工程中，单击General页签。
2. 将SDK的framework添加到Frameworks，Libraries，and Embedded Content中，并将Embed设置为Embed & Sign。



3. 单击Build Settings页签。
4. 单击Search Paths区域下的Framework Search Paths，修改为本地framework所在的目录。

常见集成问题
播放器问题

功能使用文档
基础功能

进阶功能

6.2. 基础功能
本文提供iOS播放器基础功能的使用示例。

创建播放器
本节介绍如何用最简单的方式让iOS播放器SDK播放视频，按照播放方式的不同可以分为手动播放和自动播放。

1. 创建播放器。

   创建AliPlayer播放器。

```
self.player = [[AliPlayer alloc] init];
[play setTraceID:@"xxxxxx"];        //TraceID为设备或用户的唯一标识符，通常为imei或idfa
```

2. 设置监听器。

   播放器支持设置多个监听器。

   - `prepare` 必须设置，因为手动播放需要在 `prepare` 回调中调用 `start` 开始播放。
   - `onPlayerEvent` 、 `onError` 较为重要，建议您设置。

```
@interface SimplePlayerViewController ()<AVPDelegate>
@end
- (void)viewDidLoad {
    self.player = [[AliPlayer alloc] init];
    self.player.playerView = self.avpPlayerView.playerView;
    self.player.delegate = self;
    //...
}
```

```
/**
 @brief 错误代理回调
 @param player 播放器player指针
 @param errorModel 播放器错误描述，参考AliVcPlayerErrorModel
 */
- (void)onError:(AliPlayer*)player errorModel:(AVPErrorModel *)errorModel {
    //提示错误，及stop播放
}
/**
 @brief 播放器事件回调
 @param player 播放器player指针
 @param eventType 播放器事件类型，@see AVPEventType
 */
-(void)onPlayerEvent:(AliPlayer*)player eventType:(AVPEventType)eventType {
    switch (eventType) {
        case AVPEventPrepareDone: {
            // 准备完成
        }
            break;
        case AVPEventAutoPlayStart:
            // 自动播放开始事件
            break;
        case AVPEventFirstRenderedStart:
            // 首帧显示
            break;
        case AVPEventCompletion:
            // 播放完成
            break;
        case AVPEventLoadingStart:
            // 缓冲开始
            break;
        case AVPEventLoadingEnd:
            // 缓冲完成
            break;
        case AVPEventSeekEnd:
            // 跳转完成
            break;
        case AVPEventLoopingStart:
            // 循环播放开始
            break;
        default:
            break;
    }
}
/**
 @brief 视频当前播放位置回调
 @param player 播放器player指针
 @param position 视频当前播放位置
 */
- (void)onCurrentPositionUpdate:(AliPlayer*)player position:(int64_t)position {
    // 更新进度条
}
/**
 @brief 视频缓存位置回调
```

```
 @param player 播放器player指针
 @param position 视频当前缓存位置
 */
- (void)onBufferedPositionUpdate:(AliPlayer*)player position:(int64_t)position {
    // 更新缓冲进度
}
/**
 @brief 获取track信息回调
 @param player 播放器player指针
 @param info track流信息数组 参考AVPTrackInfo
 */
- (void)onTrackReady:(AliPlayer*)player info:(NSArray<AVPTrackInfo*>*)info {
    // 获取多码率信息
}
/**
 @brief 字幕显示回调
 @param player 播放器player指针
 @param index 字幕显示的索引号
 @param subtitle 字幕显示的字符串
 */
- (void)onSubtitleShow:(AliPlayer*)player index:(int)index subtitle:(NSString *)sub
title {
    // 获取字幕进行显示
}
/**
 @brief 字幕隐藏回调
 @param player 播放器player指针
 @param index 字幕显示的索引号
 */
- (void)onSubtitleHide:(AliPlayer*)player index:(int)index {
    // 隐藏字幕
}
/**
 @brief 获取截图回调
 @param player 播放器player指针
 @param image 图像
 */
- (void)onCaptureScreen:(AliPlayer *)player image:(UIImage *)image {
    // 预览，保存截图
}
/**
 @brief track切换完成回调
 @param player 播放器player指针
 @param info 切换后的信息 参考AVPTrackInfo
 */
- (void)onTrackChanged:(AliPlayer*)player info:(AVPTrackInfo*)info {
    // 切换码率结果通知
}
```

3. 创建DataSource。
   - iOS播放器SDK支持5种点播播放方式，包括：UrlSource播放、VidAuth播放（视频点播用户推荐使用）、VidSts播放、VidMps播放、加密播放。
   - iOS播放器SDK支持2种直播播放方式，UrlSource播放和加密播放。

> **说明**
> - UrlSource是直接通过URL播放，其余的三种是通过Vid进行播放：VidSts，VidAuth（推荐）限点播用户使用；VidMps仅限媒体处理（MPS）用户使用。
> - 接入地域Region的设置，请参见 **点播地域标识**。
> - MPS视频播放的流程与概念，请参见 **视频播放**。

点播视频播放

直播UrlSource播放 直播DRM加密播放

直播视频播放

4. **设置显示View。**
   如果播放源有画面，那么需要设置显示的view到播放器中，用来显示画面。示例如下：

```
self.player.playerView = self.avpPlayerView.playerView;//用户显示的view
```

5. **（可选）开启自动播放，默认为关闭状态。**

```
self.player.autoPlay = YES;
```

6. **准备播放。**
   调用 `[self.player prepare]` **开始读取并解析数据。**

```
[self.player prepare];
```

7. **开始播放。**
   通过调用 `start` **方法开始播放视频。若创建自动播放，数据解析完成后将开始自动播放视频。**

> **注意** 自动播放的时候将不会回调 `AVPEventPrepareDone` 回调，而会回调 `AVPEventAutoPlayStart` 回调。

```
[self.player start];
```

控制播放

iOS播放器SDK支持开始、暂停、从指定时间点播放等主流操作。 指定时间播放即seek。

开始播放

开始播放视频，由 `start` 接口实现。示例如下：

```
[self.player start];
```

从指定时间开始播放

跳转到某个时刻进行播放，由 `seekToTime` 接口实现。适用于用户拖拽进度条，或续播等需要从指定时间点开始播放的场景。示例如下：

```
//posotion为指定的时间，单位：秒，目前只支持不精准模式AVP_SEEKMODE_INACCURATE。
[self.player seekToTime:position seekMode:AVP_SEEKMODE_INACCURATE];
```

暂停播放

暂停播放视频，由 `pause` 接口实现。示例如下：

```
[self.player pause];
```

停止播放

停止播放视频，由 `stop` 接口实现。示例如下：

```
[self.player stop];
```

设置显示模式

iOS播放器SDK支持填充、旋转、镜像等显示设置。

填充

支持设置宽高比适应、宽高比填充和拉伸填充这3种画面填充模式，由 `scalingMode` 接口实现。示例如下：

```
//设置宽高比适应（将按照视频宽高比等比缩小到view内部，不会有画面变形）
self.player.scalingMode = AVP_SCALINGMODE_SCALEASPECTFIT;
//设置宽高比填充（将按照视频宽高比等比放大，充满view，不会有画面变形）
self.player.scalingMode = AVP_SCALINGMODE_SCALEASPECTFILL;
//设置拉伸填充（如果视频宽高比例与view比例不一致，会导致画面变形）
self.player.scalingMode = AVP_SCALINGMODE_SCALETOFILL;
```

旋转

指画面按指定角度旋转，由 `rotateMode` 接口实现。示例如下：

```
//设置画面顺时针旋转0度
self.player.rotateMode = AVP_ROTATE_0;
//设置画面顺时针旋转90度
self.player.rotateMode = AVP_ROTATE_90;
//设置画面顺时针旋转180度
self.player.rotateMode = AVP_ROTATE_180;
//设置画面顺时针旋转270度
self.player.rotateMode = AVP_ROTATE_270;
```

镜像

支持水平镜像、垂直镜像和无镜像，由 `mirrorMode` 接口实现。示例如下：

```
//设置无镜像
self.player.mirrorMode = AVP_MIRRORMODE_NONE;
//设置水平镜像
self.player.mirrorMode = AVP_MIRRORMODE_HORIZONTAL;
//设置垂直镜像
self.player.mirrorMode = AVP_MIRRORMODE_VERTICAL;
```

获取播放信息

iOS播放器SDK支持获取当前的播放进度、播放时长和缓冲进度信息。

获取当前播放进度

指获取当前的播放时刻，需要在onCurrentPositionUpdate回调中获取position。示例如下：

```
- (void)onCurrentPositionUpdate:(AliPlayer*)player position:(int64_t)position {
//position为当前播放进度，单位为毫秒
NSString *position_ = position;
}
```

获取播放时长

指获取视频总时长。需要在视频加载完成以后才可以获取到，比如在 onPrepared 回调之后再获取。单位：毫秒。示例如下：

```
-(void)onPlayerEvent:(AliPlayer*)player eventType:(AVPEventType)eventType {
    switch (eventType) {
        case AVPEventPrepareDone: {
            if (self.player.duration >= 0) {
               NSString *duration  = self.player.duration;
            }
        }
            break;
        default:
            break;
    }
}
```

获取缓冲进度

指获取视频当前的缓冲进度，需要在onBufferedPositionUpdate回调中获取position。示例如下：

```
- (void)onBufferedPositionUpdate:(AliPlayer*)player position:(int64_t)position {
    NSString *bufferPosition = position;
}
```

监听播放状态

指监听播放器的状态，onPlayerStatusChanged回调参数为当前播放器状态。示例如下：

```
- (void)onPlayerStatusChanged:(AliPlayer*)player oldStatus:(AVPStatus)oldStatus newStat
us:(AVPStatus)newStatus {
    switch (newStatus) {
        case AVPStatusIdle:{
            //  空转，闲时，静态
        }
   break;
        case AVPStatusInitialzed:{
            //  初始化完成
        }
   break;
        case AVPStatusPrepared:{
            //  准备完成
        }
   break;
        case AVPStatusStarted:{
            //  正在播放
        }
   break;
case AVPStatusPaused:{
            //  播放暂停
        }
   break;
case AVPStatusStopped:{
            //  播放停止
        }
   break;
case AVPStatusCompletion:{
            //  播放完成
        }
   break;
case AVPStatusError:{
            //  播放错误
        }
   break;
        default:
            break;
    }
}
```

设置音量

设置音量包括音量调节和静音设置。

音量调节

指调节音量大小，由 `volume` 接口实现。设置后还可获取音量信息。示例如下：

```
//volume的值为0～1之间的实数。
self.player.volume = 1.0f;
//获取音量信息。
self.player.volume
```

静音设置

指将播放中的视频设置为静音状态，由 `muted` 接口实现。示例如下：

```
self.player.muted = YES;
```

倍速播放

iOS播放器SDK提供了倍速播放视频的功能，通过设置 `rate` 方法，能够以0.5倍~5倍速去播放视频。同时保持变声不变调。示例如下：

```
//设置倍速播放:支持0.5~5倍速的播放,通常按0.5的倍数来设置，例如0.5倍、1倍、1.5倍等
self.player.rate = 1.0f;
```

多清晰度设置

如果使用VID方式（VidAuth及VidSts）播放，无需额外设置。iOS播放器SDK会从点播服务获取清晰度列表。iOS播放器SDK支持获取和切换清晰度，UrlSource方式暂不支持此设置。

获取清晰度

当视频加载完成后，可以获取视频的清晰度。可以使用onTrackReady监听回调返回info信息，获取清晰度trackBitrate。

```
- (void)onTrackReady:(AliPlayer*)player info:(NSArray<AVPTrackInfo*>*)info {
    for (int i=0; i<info.count; i++) {
        AVPTrackInfo* track = [info objectAtIndex:i];
        switch (track.trackType) {
            case AVPTRACK_TYPE_VIDEO: {
                int trackBitrate = track.trackBitrate;
            }
                break;
        }
    }
}
```

切换清晰度

切换清晰度通过 `selectTrack` 方法，传递对应TrackInfo的index即可。

```
[self.player selectTrack:index];
```

清晰度切换通知

清晰度切换完成回调onTrackChanged。

```
- (void)onTrackChanged:(AliPlayer*)player info:(AVPTrackInfo*)info {
    // 切换完成
}
```

循环播放

iOS播放器SDK提供了循环播放视频的功能。调用 `loop` 开启循环播放，播放完成后，将会自动从头开始播放视频。示例如下：

```
self.player.loop = YES;
```

同时循环开始的回调将会使用 `AVPEventLoopingStart` 中通知。示例如下：

```
- (void)onPlayerEvent:(AliPlayer*)player eventType:(AVPEventType)eventType {
    switch (eventType) {
        case AVPEventLoopingStart:
            break;
    }
}
```

相关文档

- 接口说明
- 进阶功能

6.3. 进阶功能

本文提供iOS播放器进阶功能的使用示例。

播放

短视频列表播放

针对典型的短视频列表播放场景，iOS播放器SDK提供了完善的列表播放功能，结合预加载等机制大幅改善短视频的起播速度。长视频场景不建议使用该功能。

1. 创建播放器。

   创建AliListPlayer播放器。示例如下：

   ```
   self.listPlayer = [[AliListPlayer alloc] init];
   [listPlayer setTraceID:@"xxxxxx"];  //TraceID为设备或用户的唯一标识符，通常为imei或idfa
   ```

   2021.12.30：增加traceid透传示例代码

2. （可选）设置监听器。

   创建列表播放时，监听器为非必须配置，但如果不设置，将无法收到播放器的播放失败、播放进度等事件通知，因此，建议您设置。
   播放器支持设置多个监听器，其中 `onPlayerEvent` 、 `onError` 较为重要，建议您设置。

   ```
   /**
    @brief 错误代理回调
    @param player 播放器player指针
    @param errorModel 播放器错误描述，参考AliVcPlayerErrorModel
    */
   - (void)onError:(AliPlayer*)player errorModel:(AVPErrorModel *)errorModel {
       //提示错误，及stop播放
   }
   /**
    @brief 播放器事件回调
    @param player 播放器player指针
    @param eventType 播放器事件类型，@see AVPEventType
    */
   -(void)onPlayerEvent:(AliPlayer*)player eventType:(AVPEventType)eventType {
       switch (eventType) {
           case AVPEventPrepareDone: {
               // 准备完成
           }
               break;
           case AVPEventAutoPlayStart:
               // 自动播放开始事件break;
   ```

```
        case AVPEventFirstRenderedStart:
            // 首帧显示break;
        case AVPEventCompletion:
            // 播放完成break;
        case AVPEventLoadingStart:
            // 缓冲开始break;
        case AVPEventLoadingEnd:
            // 缓冲完成break;
        case AVPEventSeekEnd:
            // 跳转完成break;
        case AVPEventLoopingStart:
            // 循环播放开始break;
        default:
            break;
    }
}
/**
 @brief 视频当前播放位置回调
 @param player 播放器player指针
 @param position 视频当前播放位置
 */
- (void)onCurrentPositionUpdate:(AliPlayer*)player position:(int64_t)position {
    // 更新进度条
}
/**
 @brief 视频缓存位置回调
 @param player 播放器player指针
 @param position 视频当前缓存位置
 */
- (void)onBufferedPositionUpdate:(AliPlayer*)player position:(int64_t)position {
    // 更新缓冲进度
}
/**
 @brief 获取track信息回调
 @param player 播放器player指针
 @param info track流信息数组 参考AVPTrackInfo
 */
- (void)onTrackReady:(AliPlayer*)player info:(NSArray<AVPTrackInfo*>*)info {
    // 获取多码率信息
}
/**
 @brief 字幕显示回调
 @param player 播放器player指针
 @param index 字幕显示的索引号
 @param subtitle 字幕显示的字符串
 */
- (void)onSubtitleShow:(AliPlayer*)player index:(int)index subtitle:(NSString *)sub
title {
    // 获取字幕进行显示
}
/**
 @brief 字幕隐藏回调
 @param player 播放器player指针
 @param index 字幕显示的索引号
 */
```

```
 */
- (void)onSubtitleHide:(AliPlayer*)player index:(int)index {
    // 隐藏字幕
}
/**
 @brief 获取截图回调
 @param player 播放器player指针
 @param image 图像
 */
- (void)onCaptureScreen:(AliPlayer *)player image:(UIImage *)image {
    // 预览，保存截图
}
/**
 @brief track切换完成回调
 @param player 播放器player指针
 @param info 切换后的信息 参考AVPTrackInfo
 */
- (void)onTrackChanged:(AliPlayer*)player info:(AVPTrackInfo*)info {
    // 切换码率结果通知
}
//...
```

3. 设置预加载个数。

   合理设置预加载个数，能够有效的提高起播的速度。示例如下：

```
//设置预加载个数。总共加载的个数为： 1 + count*2。
self.listplyer.preloadCount = 2;
```

4. 添加或移除多个播放源。

   列表播放支持两种播放源：VidSts播放和UrlSource播放。Url为视频的播放地址，vid为点播视频的音视频ID（VideoId）。示例如下：

   - Url：播放地址可以是第三方点播地址或阿里云点播服务中的播放地址。阿里云播放地址可以调用获取音视频播放地址接口获取。建议您集成点播服务端SDK来获取音视频播放地址，免去自签名的麻烦。调用接口获取音视频播放地址的示例请参见开发者门户。
   - Vid：音视频ID，可以在音视频上传完成后通过控制台（路径：媒资库 > 音/视频。）或服务端接口（搜索媒体信息）获取。

```
//添加vid播放源
[self.listPlayer addVidSource:videoId uid:UUIDString];
//添加URL播放源
[self.listPlayer addUrlSource:URL uid:UUIDString];
//移除某个源
[self.listPlayer removeSource:UUIDString];
```

   > 说明
   >
   > - 不支持VidAuth播放和VidMps播放方式。
   > - uid是视频的唯一标志。用于区分视频是否一样。如果uid一样，则认为视频是一样的。如果播放出现串流的情况，请注意查看是不是在不同的界面设置了同一个uid。uid没有格式限制，任意字符串都行。

5. 设置显示View。

   如果播放源有画面，需要设置View到播放器中，用来显示画面。示例代码如下：

```
self.listPlayer.playerView = self.simplePlayScrollView.playView;
```

6. 播放视频源。
   添加了一个或多个播放源之后，调用 `moveTo` 便可以自动播放某个视频源。示例如下：

```
//UrlSource播放方式时使用此接口
- (BOOL) moveTo:(NSString*)uid;
//VidSts播放方式时使用此接口，需要传递STS临时凭证和临时AK对，请提前获取，获取方式请参见创建角色并进行STS临时授权。
- (BOOL) moveTo:(NSString*)uid accId:(NSString*)accId accKey:(NSString*)accKey token:(NSString*)token region:(NSString*)region;
```

7. 播放上一个、下一个视频。
   播放器提供了 `moveToPrev` ， `moveToNext` 接口用于播放上一个，下一个视频。示例如下：

   UrlSource播放源 VidSts播放源

外挂字幕

iOS播放器SDK支持添加和切换外挂字幕，现已支持SRT、SSA、ASS3种格式。示例如下：

1. 创建显示字幕的View。
   根据不同的字幕格式创建不同的View。

```
// 初始化自定义subtitleAddLab
UILabel *subTitleLabel = [[UILabel alloc] initWithFrame:frame];
// 将字幕添加至自定义的superView
[superView addSubview:subTitleLabel];
```

2. 设置字幕相关监听。

```
//  外挂字幕被添加
- (void)onSubtitleExtAdded:(AliPlayer*)player trackIndex:(int)trackIndex URL:(NSString *)URL {}
// 字幕头信息回调
- (void)onSubtitleHeader:(AliPlayer *)player trackIndex:(int)trackIndex Header:(NSString *)header {}
// 字幕显示回调
- (void)onSubtitleShow:(AliPlayer*)player trackIndex:(int)trackIndex subtitleID:(long)subtitleID subtitle:(NSString *)subtitle {
    subTitleLabel.text = subtitle;
    subTitleLabel.tag = subtitleID;
}
// 字幕隐藏回调
- (void)onSubtitleHide:(AliPlayer*)player trackIndex:(int)trackIndex subtitleID:(long)subtitleID {
    [subTitleLabel removeFromSuperview];
}
```

3. 添加字幕。

```
[self.player addExtSubtitle:URL];
```

4. 切换字幕。

```
[self.player selectExtSubtitle:trackIndex enable:YES];
```

纯音频播放

通过禁用视频播放，达到纯音频播放的效果。在prepare之前配置PlayerConfig。

```
AVPConfig *config = [self.player getConfig];
config.disableVideo = YES;
[self.player setConfig:config];
```

软硬解切换

iOS播放器SDK提供了H.264、H.265的硬解码能力，同时提供了 `enableHardwareDecoder` 提供开关。默认开，并且在硬解初始化失败时，自动切换为软解，保证视频的正常播放。示例如下：

```
//开启硬解，默认开启
self.player.enableHardwareDecoder = YES;
```

如果从硬解自动切换为软解，将会通过 `onPlayerEvent` 回调，示例如下：

```
-(void)onPlayerEvent:(AliPlayer*)player eventWithString:
(AVPEventWithString)eventWithString description:(NSString *)description {
    if (eventWithString == EVENT_SWITCH_TO_SOFTWARE_DECODER) {
        //切换到软解
    }
}
```

网络自适应切换视频清晰度

iOS播放器SDK支持多码率自适应HLS、DASH视频流。在 `prepare` 成功之后，通过 `getMediaInfo` 可以获取到各个码流的信息，即 `TrackInfo` 。示例如下：

```
AVPMediaInfo *info = [self.player getMediaInfo];
NSArray<AVPTrackInfo*>* tracks = info.tracks;
```

> **说明** HLS、DASH视频流需要经过多码率视频转码模板组打包处理，可以在点播控制台的 **配置管理** > **媒体处理配置** > **转码模板组** 中配置生产对应的视频流，详细操作请参见 **视频或字幕打包模板设置** 。

在播放过程中，可以通过调用播放器的 `selectTrack` 方法切换播放的码流，取值为SELECT_AVPTRACK_TYPE_VIDEO_AUTO时，为多码率自适应。示例如下：

```
//多码率切换
[self.player selectTrack:track.trackIndex];
//多码率切换并自适应
[self.player selectTrack:SELECT_AVPTRACK_TYPE_VIDEO_AUTO];
```

切换的结果会在 `onTrackChanged` 监听之后会回调。示例如下：

```
- (void)onTrackChanged:(AliPlayer*)player info:(AVPTrackInfo*)info {
    if (info.trackType == AVPTRACK_TYPE_VIDEO) {
        // video changed
    }
    // etc
}
```

截图

iOS播放器SDK提供了对当前视频截图的功能，由 `snapshot` 接口实现。截取的是原始的数据，并转为 `bitmap` 返回。回调接口为 `onCaptureScreen` 。示例如下：

```
//截图回调
- (void)onCaptureScreen:(AliPlayer *)player image:(UIImage *)image {
    // 处理截图
}
//截取当前播放的画面
aliyunVodPlayer.snapshot();
```

> **说明** 截图是不包含界面的。

试看

iOS播放器SDK通过配合点播服务配置，可以实现试看功能，支持VidSts和VidAuth（视频点播推荐使用此方式）两种播放方式。如何配置和使用试看功能，请参见点播试看。

配置试看功能之后，通过 `VidPlayerConfigGen` 接口的 `setPreviewTime` 方法设置播放器的试看时长。以VidSts播放方式为例，示例如下：

```
AVPVidStsSource *source = [[AVPVidStsSource alloc] init];
....
VidPlayerConfigGenerator* vp = [[VidPlayerConfigGenerator alloc] init];
[vp setPreviewTime:20];//20秒试看
source.playConfig = [vp generatePlayerConfig];//设置给播放源
...
```

当设置试看的时长，通过iOS播放器SDK播放视频时，服务端将不会返回完整的视频内容，而是返回试看时间段的内容。

> **说明** VidPlayerConfigGen支持设置服务端支持的请求参数。请参见 请求参数说明 。

设置Referer

iOS播放器SDK支持设置Referer，配合控制台的黑白名单Referer，可以控制访问权限，由 `AVPConfig` 方法设置请求Referer。iOS播放器SDK的设置示例如下：

```
//先获取配置
AVPConfig *config = [self.player getConfig];
//设置referer
config.referer = referer;
....//其他设置
//设置配置给播放器
[self.player setConfig:config];
```

设置UserAgent

iOS播放器SDK提供了 `AVPConfig` 用来设置请求UA。设置之后，播放器请求的过程中将会带上UA信息。示例如下：

```
//先获取配置
AVPConfig *config = [self.player getConfig];
//设置userAgent
config.userAgent = userAgent;
....//其他设置
//设置配置给播放器
[self.player setConfig:config];
```

配置网络重试时间和次数

支持设置iOS播放器SDK的网络超时的时间和重试次数，由 `AVPConfig` 方法实现。示例如下：

```
//先获取配置
AVPConfig *config = [self.player getConfig];
//设置网络超时时间，单位ms
config.networkTimeout = 5000;
//设置超时重试次数。每次重试间隔为networkTimeout。networkRetryCount=0则表示不重试，重试策略app决定，默认值为2
config.networkRetryCount = 2;
....//其他设置
//设置配置给播放器
[self.player setConfig:config];
```

> 说明
>
> - 如果设置了networkRetryCount：如此时发生网络问题，导致出现loading后，那么将会重试networkRetryCount次，每次的间隔时间为networkTimeout。
> - 如果重试多次之后，还是loading的状态，那么就会回调 `onError` 事件，此时AVPErrorModel.code为ERROR_LOADING_TIMEOUT。
> - 如果networkRetryCount设置为0，当网络重试超时的时候，播放器就会回调onPlayerEvent，参数eventWithString为EVENT_PLAYER_NETWORK_RETRY。此时，可以调用播放器的 `reload` 方法进行重新加载网络，或者进行其他的处理。

配置缓存和延迟控制

对于播放器来说，缓存的控制非常重要。合理的配置可以有效的加快起播速度并减少卡顿。iOS播放器SDK通过 `AVPConfig` 提供了设置缓存和延迟的控制接口。示例如下：

```
//先获取配置
AVPConfig *config = [self.player getConfig];
//最大延迟。注意：直播有效。当延时比较大时，播放器SDK内部会追帧等，保证播放器的延时在这个范围内。
config.maxDelayTime = 5000;
// 最大缓冲区时长。单位ms。播放器每次最多加载这么长时间的缓冲数据。
config.maxBufferDuration = 50000;
//高缓冲时长。单位ms。当网络不好导致加载数据时，如果加载的缓冲时长到达这个值，结束加载状态。
config.highBufferDuration = 3000;
// 起播缓冲区时长。单位ms。这个时间设置越短，起播越快。也可能会导致播放之后很快就会进入加载状态。
config.startBufferDuration = 500;
//其他设置
//设置配置给播放器
[self.player setConfig:config];
```

> **注意** 三个缓冲区时长的大小关系必须为：startBufferDuration ≤ highBufferDuration ≤ maxBufferDuration。

设置HTTP Header

通过 `AVPConfig` 方法，可以给播放器中的请求加上HTTP的header参数。示例如下：

```
//先获取配置
AVPConfig *config = [self.player getConfig];
//定义header
NSMutableArray *httpHeaders = [[NSMutableArray alloc] init];
//比如使用httpdns时，需要设置Host。
[httpHeaders addObject:@"Host:example.com"];
//设置header
config.httpHeaders = httpHeaders;
....//其他设置
//设置配置给播放器
[self.player setConfig:config];
```

性能

本地缓存

iOS播放器SDK提供了本地缓存（边播边缓存）的功能，能够让用户重复播放视频时，达到省流量的目的。只需在 `prepare` 之前给播放器配置 `AVPCacheConfig` 即可实现此功能。示例如下：

```
AVPCacheConfig *config = [[AVPCacheConfig alloc] init];
//开启缓存功能
config.enable = YES;
//能够缓存的单个文件最大时长。超过此长度则不缓存
config.maxDuration = 100;
//缓存目录的位置，需替换成app期望的路径
config.path = @"please use your cache path here";
//缓存目录的最大大小。超过此大小，将会删除最旧的缓存文件
config.maxSizeMB = 200;
//设置缓存配置给到播放器
[self.player setCacheConfig:config];
```

缓存成功之后，以下情况将会利用缓存文件（必须已经设置了 `setCacheConfig` ）

- 如果设置了循环播放，即 `loop(YES)` ，那么第二次播放的时候，将会自动播放缓存的文件。
- 缓存成功后，重新创建播放器，播放同样的资源，也会自动使用缓存文件。

> **说明** vid的缓存文件是通过vid等信息定位的，所以对于vid的缓存文件，将需要网络请求信息之后才能确定使用哪个缓存文件。

iOS播放器SDK提供了获取缓存文件路径的接口：

API接口	描述	参数	返回值
(NSString *) getCacheFilePath:(NSString *)URL	根据URL获取缓存的文件名。必须先调用setCacheConfig才能获取到。	URL	最终缓存的文件绝对路径。

API接口	描述	参数	返回值
(NSString *) getCacheFilePath:(NSString *)vid format:(NSString *)format definition: (NSString *)definition	根据vid获取缓存的文件名。	• vid：视频ID • format：视频格式 • definition：视频清晰度 • previewTime：试看时长	最终缓存的文件绝对路径。

边播边缓存的使用有限制条件，具体如下：

- 对于UrlSource播放方式。如果是HLS（即M3U8）地址，将不会缓存。如果是其他支持的格式，则根据缓存配置进行缓存。
- 对于VidAuth、VidSts播放方式，将会根据缓存配置进行缓存。
- 播放器读取完全部的数据则视为缓存成功。如果在此之前，调用 `stop` ，或者出错 `onError` ，则缓存将会失败。
- cache内的seek的操作不会影响缓存结果。cache外的seek会导致缓存失败。
- 如果视频源是加密的，此时如果加密文件与App信息不一致，将会缓存失败。
- cache的结果回调，会通过 `onPlayerEventInfo` 回调。

```
-(void)onPlayerEvent:(AliPlayer*)player eventWithString:
(AVPEventWithString)eventWithString description:(NSString *)description {
    if (eventWithString == EVENT_PLAYER_CACHE_SUCCESS) {
        //缓存成功事件。
    }else if (eventWithString == EVENT_PLAYER_CACHE_ERROR) {
        //缓存失败事件。
    }
}
```

预加载

iOS播放器SDK提供预加载功能，是对本地缓存（边播边缓存）功能的升级，通过设置视频缓存的内存占用大小，更能提升视频的起播速度。

预加载功能的使用说明如下所示：

- 目前支持MP4、MP3、FLV、HLS（单码率视频流）等单个媒体文件的加载。
- 仅支持UrlSource播放方式播放视频的预加载，暂不支持VidAuth、VidSts方式播放视频的预加载。

1. 开启本地缓存功能。

   本地缓存功能默认关闭，如需使用，需要手动开启。通过AliPlayerGlobalSettings中的 `enableLocalCache` 控制。

```
/**
 * 开启本地缓存，开启之后，就会缓存到本地文件中。
 * @param enable：本地缓存功能开关。true：开启，false：关闭，默认关闭。
 * @param maxBufferMemoryKB：必须设置，设置能够缓存的单个视频源文件的最大内存占用大小，超过此大小则不缓存，单位：KB。
 * @param localCacheDir：必须设置，本地缓存的文件目录，为绝对路径。
 */
[AliPlayerGlobalSettings enableLocalCache:true maxBufferMemoryKB:1024
localCacheDir:@""];
/**
 @brief 本地缓存文件自动清理相关的设置
 @param expireMin：设置缓存的过期时间。单位：分钟，默认值30天，过期的缓存不管容量多大，都会被删除。
 @param maxCapacityMB：最大缓存容量。单位：兆，默认值20GB，在清理时，如果缓存总容量超过此大小，则会以cacheItem为粒度，按缓存的最后时间排序，一个一个的删除最旧的缓存文件，直到小于等于最大缓存容量。
 @param freeStorageMB：磁盘最小空余容量。单位：兆，默认值0，在清理时，同最大缓存容量，如果当前磁盘容量小于该值，也会按规则一个一个的删除缓存文件，直到freeStorage大于等于该值或者所有缓存都被清理掉。
 */
[AliPlayerGlobalSettings setCacheFileClearConfig:0 maxCapacityMB:0
freeStorageMB:0];
/**
 * 获取加载url的hash值回调，用来做url唯一的id，必须要保证每个url都不一样
 * 其中block为 typedef NSString *(*CaheUrlHashCallback)(NSString *url);
 */
// 需要自己实现这个函数并把函数指针交给setCacheUrlHashCallback
static NSString *CaheUrlHashHandle(NSString *url) {
    return @"xxx";
}
[AliPlayerGlobalSettings setCacheUrlHashCallback:&CaheUrlHashHandle];
```

> **说明**
>
> - 如果视频播放URL带有鉴权参数，预加载和播放时鉴权参数会变化，可以将URL的鉴权参数去掉后再计算md5值。例如：带有鉴权参数的视频播放URL为 `http://****.mp4?aaa` ，则加载时使用 `http://****.mp4` 计算md5值。
> - 如果服务器同时支持HTTP和HTTPS协议，但是不同的协议指向的媒体文件是同一个，则可以将请求头去掉或者统一后再计算md5值。例如：
>   - 视频播放URL为 `https://****.mp4` 和 `http://****.mp4` ，则加载时使用 `***.mp4` 计算md5值。
>   - 视频播放URL为 `https://****.mp4` ，加载时统一为 `http://****.mp4` 后再计算md5值。

2. 获取AliMediaLoader实例。

   AliMediaLoader实例为单例，即无论获取多少次，创建的都是同一个实例。

```
[AliMediaLoader shareInstance];
```

3. 配置AliMediaLoader。

```
//配置回调，并开始加载。
// url：为视频文件地址。 duration：加载的时长大小，单位：毫秒。
[[AliMediaLoader shareInstance] load:obj.url duration:1000];
// 设置回调代理
[[AliMediaLoader shareInstance] setAliMediaLoaderStatusDelegate:self];
@protocol AliMediaLoaderStatusDelegate <NSObject>
@optional
/**
 @brief 错误回调
 @param url 加载url
 @param code 错误码
 @param msg 错误描述
 */
- (void)onError:(NSString *)url code:(int64_t)code msg:(NSString *)msg;
/**
 @brief 完成回调
 @param url 加载url
 */
- (void)onCompleted:(NSString *)url;
/**
 @brief 取消回调
 @param url 加载url
 */
- (void)onCanceled:(NSString *)url;
@end
```

4. （可选）删除加载文件。

   可按需删除加载文件，以节省空间。iOS播放器SDK不提供删除接口，需要在App删除加载目录下的文件。

获取下载速度

指获取当前播放视频的下载速度，在onCurrentDownloadSpeed回调中获取speed。示例如下：

```
- (void)onCurrentDownloadSpeed:(AliPlayer *)player speed:(int64_t)speed {
    int speed_ = speed;
}
```

网络特性
使用HTTP/2

iOS播放器支持使用HTTP/2协议，该协议通过多路复用，避免队头阻塞，以改善播放性能。示例如下：

```
[AliPlayerGlobalSettings setUseHttp2:true];
```

视频下载

iOS播放器SDK提供了点播服务视频的下载功能，能够通过VidSts和VidAuth下载点播服务上的视频。同时，下载的方式提供了安全下载和普通下载的模式（可登录点播控制台，选择**配置管理** > **分发加速配置** > **下载设置配置**）。

- 普通下载：下载后的视频数据未经过阿里云加密，用户可以用第三方播放器播放。
- 安全下载：下载后的视频数据经过阿里云加密。第三方播放器无法播放。仅支持使用阿里云的播放器进行播放。

1. 创建并设置下载器。

创建下载器。示例如下：

```
AliMediaDownloader *downloader = [[AliMediaDownloader alloc] init];
[downloader setSaveDirectory:self.downLoadPath];
[downloader setDelegate:self];
```

下载SDK支持私有加密的下载。为了保证安全性，需要配置一个加密校验文件到播放器。（建议在Application中配置一次即可）。示例如下：

```
NSString *encrptyFilePath = [[NSBundle mainBundle] pathForResource:@"encryptedApp"
ofType:@"dat"];
[AliPrivateService initKey:encrptyFilePath];
```

> 注意 如果是安全下载，并且加密校验文件与App信息不一致，会导致下载失败。

2. 设置监听事件。
下载器提供了多个事件监听。示例如下：

```
-(void)onPrepared:(AliMediaDownloader *)downloader mediaInfo:(AVPMediaInfo *)info {
    //准备下载项成功
}
-(void)onError:(AliMediaDownloader *)downloader errorModel:(AVPErrorModel
*)errorModel {
    //下载出错
}
-(void)onDownloadingProgress:(AliMediaDownloader *)downloader percentage:
(int)percent {
    //下载进度百分比
}
-(void)onProcessingProgress:(AliMediaDownloader *)downloader percentage:
(int)percent {
    //处理进度百分比
}
-(void)onCompletion:(AliMediaDownloader *)downloader {
    //下载成功
}
```

3. 准备下载源。
通过 `preapre` 方法准备下载源。下载源支持AVPVidStsSource和AVPVidAuthSource两种。以AVPVidStsSource举例，示例如下：

```
//创建VidSts
AVPVidStsSource* stsSource = [[AVPVidStsSource alloc] init];
stsSource.vid = source.vid;//视频vid
stsSource.region = DEFAULT_SERVER.region;//接入区域
stsSource.securityToken = DEFAULT_SERVER.securityToken;//安全token
stsSource.accessKeySecret = DEFAULT_SERVER.accessKeySecret;//临时akSecret
stsSource.accessKeyId = DEFAULT_SERVER.accessKeyId;//临时akId
//准备下载源
[downloader prepareWithVid:stsSource];
```

4. 准备成功后选择下载项。
准备成功后，会回调 `onPrepared` 方法。选择某个Track进行下载：

```
-(void)onPrepared:(AliMediaDownloader *)downloader mediaInfo:(AVPMediaInfo *)info {
    NSArray<AVPTrackInfo*>* tracks = info.tracks;
    //比如：下载第一个TrackInfo
    [downloader selectTrack:[tracks objectAtIndex:0].trackIndex];
}
```

5. 更新下载源并开始下载。
   为了防止VidSts和VidAuth过期，建议更新下载源的信息后开始下载。示例如下：

```
//更新下载源
[downloader updateWithVid:vidSource]
//开始下载
[downloader start];
```

6. 下载成功或失败后，释放下载器。
   下载成功后，调用 `destroy` 释放下载器。

```
[self.downloader destroy];
self.downloader = nil;
```

视频加密播放

点播视频支持HLS标准加密、阿里云私有加密和DRM加密，直播视频仅支持DRM加密。加密播放请参见 视频加密播放。

Native RTS播放

iOS播放器SDK集成Native RTS SDK实现Native端低延时直播功能，详情请参见 阿里云播放器SDK集成Native RTS SDK实现说明（iOS端）。

异常处理

使用阿里云播放器播放视频发生异常时，可借助单点探查功能快速定位问题，详细内容，请参见 单点探查。2022.01.28：新增单点探查功能，补充相关说明。

相关文档

接口说明

6.4. 接口说明

本文提供iOS播放器SDK不同版本的接口文档链接。

SDK版本	链接
V5.4.5.0	iOS播放器SDK接口文档（V5.4.5.0）
V5.4.4.0	iOS播放器SDK接口文档（V5.4.4.0）
V5.4.2.0	iOS播放器SDK接口文档（V5.4.2.0）
V5.4.1	iOS播放器SDK接口文档（V5.4.1）
V5.4.0	iOS播放器SDK接口文档（V5.4.5.0）
V5.3.2	iOS播放器SDK接口文档（V5.3.2）
V5.3.0	iOS播放器SDK接口文档（V5.3.0）
V5.2.2	iOS播放器SDK接口文档（V5.2.2）

SDK版本	链接
V5.2.1	iOS播放器SDK接口文档（V5.2.1）
V5.1.6	iOS播放器SDK接口文档（V5.1.6）
V5.1.5	iOS播放器SDK接口文档（V5.1.5）

7.Flutter播放器

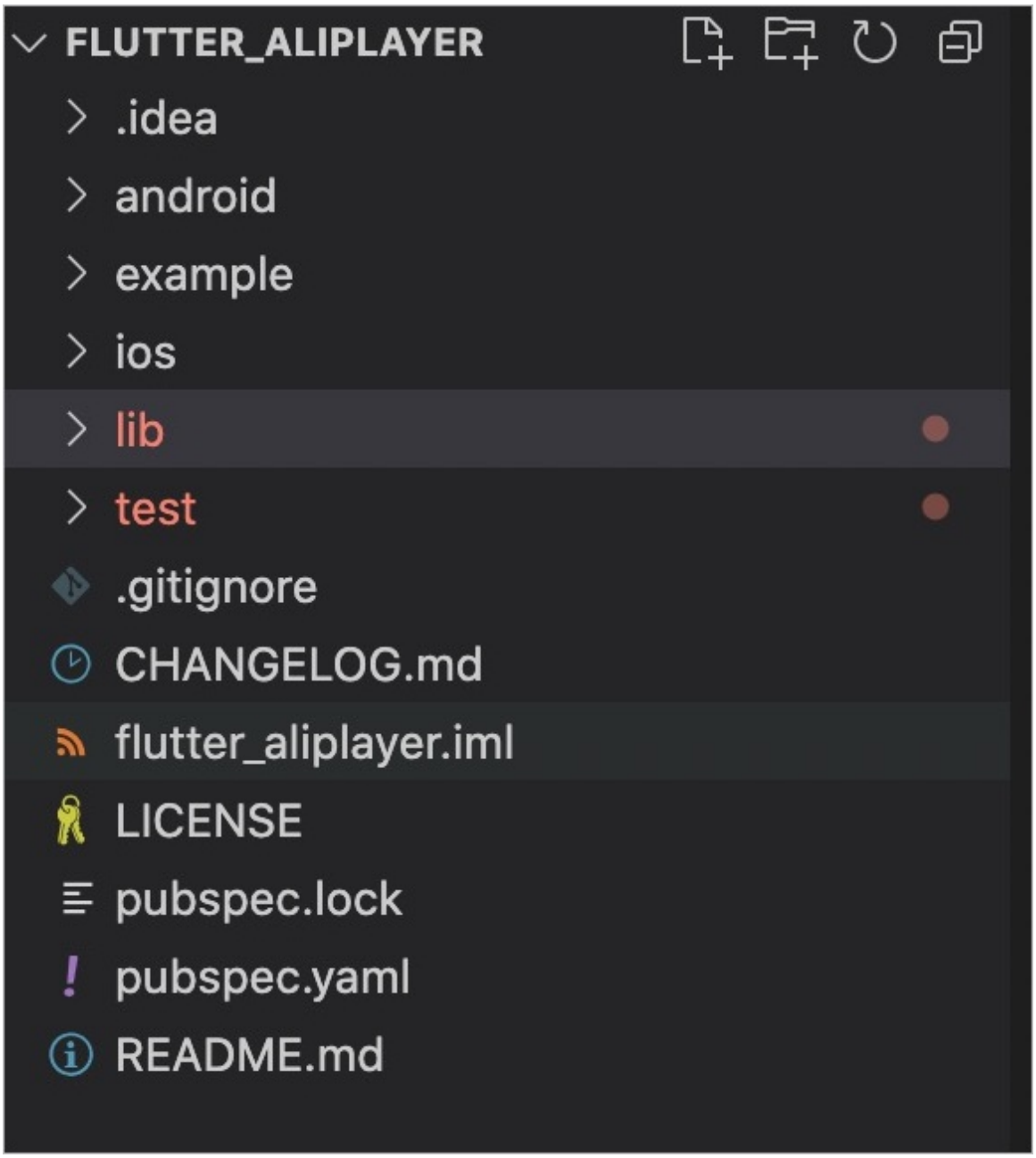
7.1. 快速集成

本文为您介绍了Flutter播放器SDK的说明以及集成方式。

项目说明

Flutter播放器SDK原生层的开发基于Android播放器SDK和iOS播放器SDK。目前已将源码通过Demo依赖的方式透出，开发者可以自行添加。最新版Flutter播放器SDK源码请参见SDK简介与下载。

项目目录结构如图所示：



目录文件名及功能说明如下表所示。

文件名	内容	是否必需
android	Android端原生代码与播放器SDK。	是
ios	iOS端原生代码与播放器SDK。	是
lib	Flutter端接口代码。	是
example	Flutter播放器Demo。	否

使用说明

Android播放器SDK不支持模拟器，集成完成后需要真机运行。

集成方式

您可以通过集成依赖的方式，在项目中的 *pubspec.yaml*文件中引入flutter_aliplayer依赖，快速集成Flutter播放器。

```
dependencies:
  flutter_aliplayer: ^version
# version为阿里云Flutter播放器的版本号，例如：5.4.0。在使用时，请根据使用的版本进行变更。
```

集成操作

环境准备

安装Flutter：下载地址，请参见Flutter下载。具体版本支持情况请参见 Flutter播放器SDK。

集成Flutter SDK

> **说明** 以下步骤以1.65.2版本VS Code开发工具为例，其余开发工具的操作步骤类似。

1. 启动VS Code开发工具。
2. 选择**View** > **Command Palette…** 。
3. 输入*flutter*，然后选择**Flutter: New Project**。
4. 输入Project名称（如myapp），然后按回车键。
5. 指定放置项目的位置，然后单击蓝色的确定按钮。

   等待项目创建，直到显示*main.dart*文件即项目创建完成。
6. 在*pubspec.yaml*文件中引入flutter_aliplayer依赖。

   ```
   dependencies:
     flutter_aliplayer: ^version
   # version为阿里云Flutter播放器的版本号，例如：5.4.0。在使用时，请根据使用的版本进行变更。
   dependencies:
     flutter_aliplayer: ^version
   # version为阿里云Flutter播放器的版本号，例如：5.4.0。在使用时，请根据使用的版本进行变更。
   ```
7. 如果业务需要支持RTS低时延直播，请引用以下依赖。如需了解更多RTS详细信息，请参见 低延时直播简介。 2Flutter 5.4.0（上述示例版本）如需使用RTS，是否就是对应flutter_aliplayer_artc: ^5.2.3，flutter_aliplayer_rts: ^1.5.0版本号；每次版本变更版本号都需更新，开发建议无需指定具体版本号，只提供给用户版本号的查看路径即可，此处已添加说明

   > **说明** RTS和Player播放器的SDK版本可以通过 `flutter_aliplayer` 项目中的*CHANGELOG.md*获取。

```
 flutter_aliplayer_artc: ^version
 flutter_aliplayer_rts: ^version
#  version为版本号。例如，flutter_aliplayer_artc: ^5.2.3， flutter_aliplayer_rts: ^1.5.0。在使用时，请根据使用的版本进行变更。
```

下面废弃
当您集成完依赖之后，可以使用以下代码，快速搭建Flutter播放器。

1. **在*lib*文件夹下新建一个Dart File，创建并配置播放器。 1.下方代码中哪些参数是用户需要自己修改的，哪些是必填的 2.是否支持其他播放方式，如STS、MPS等，如果支持，需要如何播放，需要提供一下示例代码；；此处问题在功能使用中对代码中的参数已有详细介绍，此处只做说明，保证用户复制代码后能正常使用**

> **说明** 此处代码只做示例说明，如需了解代码中涉及到的参数详情或其他播放方式，请参见 **常用功能**。

```
import 'package:flutter/material.dart';
import 'package:flutter_aliplayer/flutter_aliplayer.dart';
import 'package:flutter_aliplayer/flutter_aliplayer_factory.dart';
class MyTest extends StatefulWidget {
  const MyTest({Key? key}) : super(key: key);
  @override
  _MyTestState createState() => _MyTestState();
}
class _MyTestState extends State<MyTest> {
  late FlutterAliplayer fAliplayer;
  @override
  void initState() {
    super.initState();
    //创建播放器
    fAliplayer = FlutterAliPlayerFactory.createAliPlayer();
  }
  //设置渲染的 View
  @override
  Widget build(BuildContext context) {
    var x = 0.0;
    var y = 0.0;
    Orientation orientation = MediaQuery.of(context).orientation;
    var width = MediaQuery.of(context).size.width;
    var height;
    if (orientation == Orientation.portrait) {
      height = width * 9.0 / 16.0;
    } else {
      height = MediaQuery.of(context).size.height;
    }
    AliPlayerView aliPlayerView = AliPlayerView(
        onCreated: onViewPlayerCreated,
        x: x,
        y: y,
        width: width,
        height: height);
    return OrientationBuilder(
      builder: (BuildContext context, Orientation orientation) {
        return Scaffold(
```

```
            body: Column(
              children: [
                Container(
                    color: Colors.black,
                    child: aliPlayerView,
                    width: width,
                    height: height),
              ],
            ),
          );
        },
      );
    }
    void onViewPlayerCreated(viewId) async {
      //将渲染View设置给播放器
      fAliplayer.setPlayerView(viewId);
      //设置播放源，URL播放方式
      fAliplayer.setUrl("https://vidoe-****/test-****.mp4");
      //开启自动播放
      fAliplayer.setAutoPlay(true);
      fAliplayer.prepare();
    }
}
```

注意 上述代码中设置的播放地址的URL是无效的URL，请根据您的实际情况使用有效的播放地址替换该URL再进行播放。

2. 在*lib*文件夹下选择跳转的dart文件，引入Flutter播放器的方法。本文以 `main.dart` 文件中跳转为例。

   跳转示例代码，仅供参考。

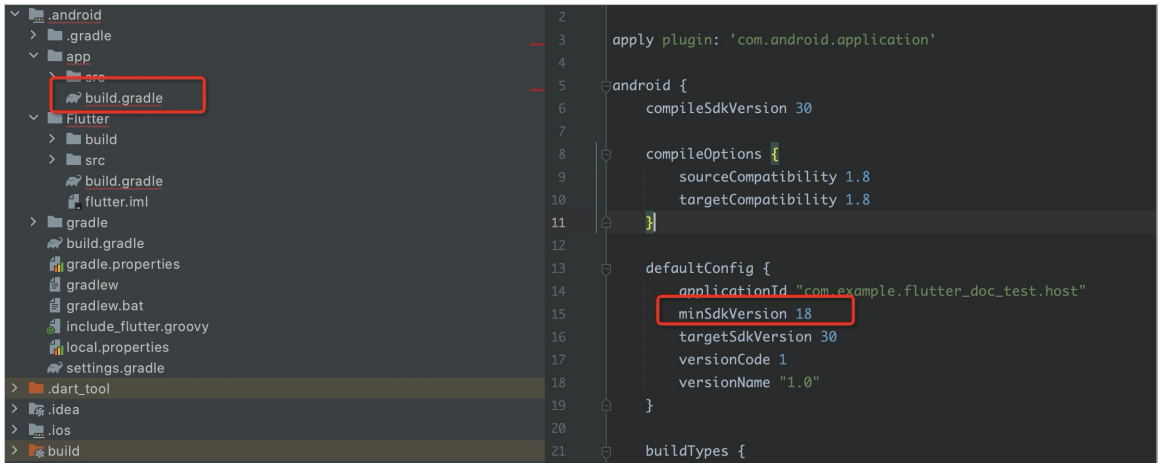
   ```
   Navigator.push(context, MaterialPageRoute(builder: (context) => MyTest())); //MyTes
   t()为方法名称，可根据名称自行变更。
   ```

3. 在上方菜单栏选择真机，单击 **Run**按钮运行。

   如果单击Run后提示如下错误信息： `uses-sdk:minSdkVersion 16 cannot be smaller than versio n 18 declared in library [:flutter_aliplayer]` ，需要将*android/app/build.gradle*以及*android/Flu tter/build.gradle*文件中的*minSdkVersion*的值修改为大于等于18，如下图所示。

   android/app/build.gradle

android/Flutter/build.gradle



7.2. 常用功能

本文提供Flutter播放器SDK常用功能的使用示例。

基础功能

创建播放器

本节介绍如何用最简单的方式让Flutter播放器SDK播放视频，按照播放方式的不同可以分为手动播放和自动播放。

1. 创建播放器。

   创建单实例播放器。

   ```
   FlutterAliplayer fAliplayer = FlutterAliPlayerFactory.createAliPlayer();
   ```

   创建多实例播放器。
   需要在Flutter层管理playerId，在播放器的回调中会返回对应的playerId，用来通知Flutter层。

   ```
   FlutterAliplayer fAliplayer = FlutterAliPlayerFactory.createAliPlayer(playerId: pla
   yerId);
   ```

2. 设置监听器。

   播放器支持设置多个监听器。

   - 创建手动播放时，`OnPrepard`必须设置，因为手动播放需要在`OnPrepard`回调中调用`play`开始播放。
   - `OnTrackReady`、`OnError`较为重要，建议您设置。

   以下示例仅展示部分接口，如下所示：

```
//准备成功
fAliplayer.setOnPrepard((playerId) {});
//首帧显示
fAliplayer.setOnRenderingStart((playerId) {});
//视频宽高变化
fAliplayer.setOnVideoSizeChanged((width, height,playerId) {});
//播放器状态变化
fAliplayer.setOnStateChanged((newState,playerId) {});
//加载状态
fAliplayer.setOnLoadingStatusListener(
    loadingBegin: (playerId) {},
    loadingProgress: (percent, netSpeed,playerId) {},
    loadingEnd: (playerId) {});
//拖动完成
fAliplayer.setOnSeekComplete((playerId) {});
//播放器事件信息回调，包括buffer、当前播放进度等等信息，根据infoCode来判断，对应FlutterAvpdef.
infoCode
fAliplayer.setOnInfo((infoCode, extraValue, extraMsg,playerId) {});
//播放完成
fAliplayer.setOnCompletion((playerId) {});
//设置流准备完成
fAliplayer.setOnTrackReady((playerId) {});
//截图结果
fAliplayer.setOnSnapShot((path,playerId) {});
//错误结果
fAliplayer.setOnError((errorCode, errorExtra, errorMsg,playerId) {});
//切换流变化
fAliplayer.setOnTrackChanged((value,playerId) {});
```

3. 创建播放源。
   - Flutter播放器SDK支持5种点播播放方式，包括：UrlSource播放、VidAuth播放（视频点播用户推荐使用）、VidSts播放、VidMps播放、加密播放。
   - Flutter播放器SDK仅支持1种直播播放方式，为UrlSource播放。

> 注意 请检查您填写字典的Key，确保字典Key与此处配置的Key名称一致，否则可能导致播放器SDK无法获取您的数据。

点播视频播放

点播UrlSource播放 | VidAuth播放 | 点播VidSts播放 | MPS播放 | 点播加密播放

直播视频UrlSource播放

使用UrlSource播放方式播放直播视频，需要将播放器的setUrl属性设置为直播拉流地址。播放地址可以是第三方直播地址或阿里云直播服务中的拉流地址。阿里云直播拉流地址可以通过直播控制台的地址生成器生成。详情请参见阿里云直播地址生成器 。

```
void onViewPlayerCreated(viewId) async {
  ///将渲染的View设置给播放器
  fAliplayer.setPlayerView(viewId);
  //设置播放源
  switch (_playMode) {
    //UrlSource播放方式
    case ModeType.URL:
      this.fAliplayer.setUrl("填写资源的播放地址"); //播放地址可以是第三方直播地址，或阿里云直播服务中的拉流地址。
      break;
    this.fAliplayer.setVidMps(authInfo);
    default:
  }
}
```

4. 设置显示View。

   如果播放源有画面，那么需要设置显示的view到播放器中，用来显示画面。示例如下：

```
@override
Widget build(BuildContext context) {
  var x = 0.0;
  var y = 0.0;
  Orientation orientation = MediaQuery.of(context).orientation;
  var width = MediaQuery.of(context).size.width;
  var height;
  if (orientation == Orientation.portrait) {
    height = width * 9.0 / 16.0;
  } else {
    height = MediaQuery.of(context).size.height;
  }
  AliPlayerView aliPlayerView = AliPlayerView(
      onCreated: onViewPlayerCreated,
      x: x,
      y: y,
      width: width,
      height: height);
  return OrientationBuilder(
    builder: (BuildContext context, Orientation orientation) {
      return Scaffold(
        body: Column(
          children: [
            Container(
                color: Colors.black,
                child: aliPlayerView,
                width: width,
                height: height),
          ],
        ),
      );
    },
  );
```

5. （可选）开启自动播放，默认为关闭状态。

```
fAliplayer.setAutoPlay(true);
```

6. 准备播放。

   调用 `prepare()` 方法准备播放。

```
fAliplayer.prepare();
```

7. 开始播放。
   - 如果未开启自动播放，需要在 `OnPrepard` 回调中调用 `fAliplayer.play();` 开始播放视频。
   - 如果开启了自动播放，则不需要调用 `fAliplayer.play();` ，数据解析完成后将开始自动播放视频。

```
fAliplayer.play();
```

控制播放

Flutter播放器SDK支持开始、暂停、从指定时间点播放等主流操作。 指定时间播放即seek。

开始播放

开始播放视频，由 `play` 接口实现。示例如下：

```
 fAliplayer.play();
```

从指定时间开始播放

跳转到某个时刻进行播放，由 `seek` 接口实现。适用于用户拖拽进度条，或续播等需要从指定时间点开始播放的场景。示例如下：

```
////posotion为指定的时间，单位：秒。seekMode取值：FlutterAvpdef.ACCURATE（精准seek）和FlutterAvpdef.INACCURATE（非精准seek）
fAliplayer.seek(position,seekMode);
```

暂停播放

暂停播放视频，由 `pause` 接口实现。示例如下：

```
fAliplayer.pause();
```

停止播放

停止播放视频，由 `stop` 接口实现。示例如下：

```
fAliplayer.stop();
```

设置显示模式

Flutter播放器SDK支持填充、旋转、镜像等显示设置。

填充

支持设置宽高比适应、宽高比填充和拉伸填充这3种画面填充模式，由 `setScalingMode` 接口实现。示例如下：

```
//设置宽高比适应（将按照视频宽高比等比缩小到view内部，不会有画面变形）
fAliplayer.setScalingMode(FlutterAvpdef.AVP_SCALINGMODE_SCALEASPECTFIT);
//设置宽高比填充（将按照视频宽高比等比放大，充满view，不会有画面变形）
fAliplayer.setScalingMode(FlutterAvpdef.AVP_SCALINGMODE_SCALEASPECTFILL);
//设置拉伸填充（如果视频宽高比例与view比例不一致，会导致画面变形）
fAliplayer.setScalingMode(FlutterAvpdef.AVP_SCALINGMODE_SCALETOFILL);
```

旋转

画面按指定角度旋转，由 `setRotateMode` 接口实现。设置后还可查询旋转角度。示例如下：

```
//设置画面顺时针旋转0度
fAliplayer.setRotateMode(FlutterAvpdef.AVP_ROTATE_0);
//设置画面顺时针旋转90度
fAliplayer.setRotateMode(FlutterAvpdef.AVP_ROTATE_90);
//设置画面顺时针旋转180度
fAliplayer.setRotateMode(FlutterAvpdef.AVP_ROTATE_180);
//设置画面顺时针旋转270度
fAliplayer.setRotateMode(FlutterAvpdef.AVP_ROTATE_270);
//获取旋转角度
fAliplayer.getRotateMode();
```

镜像

支持水平镜像、垂直镜像和无镜像，由 `setMirrorMode` 接口实现。示例如下：

```
//设置无镜像
fAliplayer.setMirrorMode(FlutterAvpdef.AVP_MIRRORMODE_NONE);
//设置水平镜像
fAliplayer.setMirrorMode(FlutterAvpdef.AVP_MIRRORMODE_HORIZONTAL);
//设置垂直镜像
fAliplayer.setMirrorMode(FlutterAvpdef.AVP_MIRRORMODE_VERTICAL);
```

获取播放信息

Flutter播放器SDK支持获取当前的播放进度和播放时长。

获取当前播放进度

获取当前的播放时刻，在onInfo回调中获取，单位毫秒。示例如下：

```
fAliplayer.setOnInfo((infoCode, extraValue, extraMsg, playerId) {
        if (infoCode == FlutterAvpdef.CURRENTPOSITION) {
            //extraValue为当前播放进度
    }
});
```

获取播放时长

获取视频总时长。需要在视频加载完成以后才可以获取到，可以在AVPEventPrepareDone事件后获取duration。示例如下：

```
fAliplayer.getMediaInfo().then((value) {
        _videoDuration = value['duration'];
});
```

监听播放状态

监听播放器的状态，onStateChanged 回调参数为当前播放器状态。示例如下：

```
fAliplayer.setOnStateChanged((newState, playerId) {
    //newState 为播放状态
  switch (newState) {
            case FlutterAvpdef.AVPStatus_AVPStatusIdle:         //  空转、闲时、静态
                    break;
            case FlutterAvpdef.AVPStatus_AVPStatusInitialzed:   //  初始化完成
                    break;
            case FlutterAvpdef.AVPStatus_AVPStatusPrepared:     //  准备完成
                    break;
            case FlutterAvpdef.AVPStatus_AVPStatusStarted:      //  正在播放
                    break;
            case FlutterAvpdef.AVPStatus_AVPStatusPaused:       //  播放暂停
          break;
            case FlutterAvpdef.AVPStatus_AVPStatusStopped:      //  播放停止
          break;
            case FlutterAvpdef.AVPStatus_AVPStatusCompletion:   //  播放完成
          break;
            case FlutterAvpdef.AVPStatus_AVPStatusError:        //  播放错误
          break;
        default:
    }
});
```

设置音量

设置音量包括音量调节和静音设置。

音量调节

调节音量大小，由 `setVolume` 接口实现。设置后还可获取音量信息。示例如下：

```
//volume的值为0～1之间的实数。
fAliPlayer.setVolume(1);
//获取音量信息。
fAliPlayer.getVolume();
```

静音设置

将播放中的视频设置为静音状态，由 `setMute` 接口实现。示例如下：

```
fAliplayer.setMute(true);
```

倍速播放

Flutter播放器SDK提供了倍速播放视频的功能，通过设置 `setRate` 方法，能够以0.5倍~5倍速去播放视频。同时保持变声不变调。示例如下：

```
//设置倍速播放：支持0.5～5倍速的播放，通常按0.5的倍数来设置，例如0.5倍、1倍、1.5倍等
fAliplayer.setRate(1.0);
```

多清晰度设置

如果使用VID方式（VidAuth（推荐）及VidSts）播放，无需额外设置。Flutter播放器SDK会从点播服务获取清晰度列表。Flutter播放器SDK支持获取和切换清晰度，UrlSource播放方式暂不支持此设置。

获取清晰度

当视频加载完成后，可以获取视频的清晰度。

```
fAliplayer.getMediaInfo().then((value) {
        AVPMediaInfo info = AVPMediaInfo.fromJson(value);
    info.tracks.forEach((element) {
                if(element.trackType == 3){
                //清晰度
              String definition = element.trackDefinition;
            //流索引
            int index = element.trackIndex;
                }
        }
});
```

切换清晰度

通过 `selectTrack` 方法切换清晰度，传递对应TrackInfo的index即可。

```
fAliplayer.selectTrack(trackIdx);
```

清晰度切换通知

清晰度切换成功回调。

```
fAliplayer.setOnTrackChanged((value, playerId) {
        //有回调表示切换成功，切换失败接口暂未提供
});
```

循环播放

Flutter播放器SDK提供了循环播放视频的功能。调用 `setLoop` 开启循环播放，播放完成后，将会自动从头开始播放视频。示例如下：

```
fAliplayer.setLoop(true);
```

同时循环开始的回调将会使用 `onInfo` 通知。示例如下：

```
fAliplayer.setOnInfo((infoCode, extraValue, extraMsg, playerId) {
      if(infoCode == FlutterAvpdef.LOOPINGSTART){
        //循环播放开始通知
      }
});
```

进阶功能

短视频列表播放

1. 创建列表播放器。

```
FlutterAliListPlayer fAliListPlayer =
FlutterAliPlayerFactory.createAliListPlayer();
```

2. 添加资源、移除资源。

   列表播放器目前只支持两种播放方式：UrlSource播放和VidSts播放。

```
//uid是视频的唯一标志。用于区分视频是否一样。如果uid一样，则认为是一样的
fAliListPlayer.addUrlSource(url,uid);
fAliListPlayer.addVidSource(vid,uid);
fAliListPlayer.removeSource(uid);
```

3. 设置预加载个数。

   合理设置预加载个数，能够有效的提高起播的速度。示例如下：

```
//设置预加载个数。总共加载的个数为： 1 + count*2。
fAliListPlayer.setPreloadCount(count);
```

4. 播放视频源。

```
//uid为必填项，如果是URL播放方式，只需要uid即可，如果是STS方式，则需要填写STS信息
fAliListPlayer.moveTo();
```

软硬解切换

Flutter播放器SDK提供了H.264、H.265的硬解码能力，同时提供了 `setEnableHardwareDecoder` 开关。默认开，并且在硬解初始化失败时，自动切换为软解，保证视频的正常播放。示例如下：

```
//开启硬解，默认开启
fAliplayer.setEnableHardwareDecoder(enable);
```

网络自适应切换视频清晰度

Flutter播放器SDK支持多码率自适应HLS、DASH视频流。在 `prepare` 成功之后，通过 `getMediaInfo` 可以获取到各个码流的信息，即 `TrackInfo` 。示例如下：

```
fAliplayer.getMediaInfo().then((value) {
//value为map，value['tracks']可以获取对应的TrackInfos 列表信息，可以参考Demo中AVPMediaInfo in
fo = AVPMediaInfo.fromJson(value); 了解如何解析TrackInfo
};
```

在播放过程中，可以通过调用播放器的 `selectTrack` 方法切换播放的码流，参数为 `TrackInfo` 中的 `trackIndex` ，切换的结果会在 `OnTrackChangedListener` 监听之后会回调。

```
//多码率切换
fAliplayer.selectTrack(index);
//多码率切换并自适应
fAliplayer.selectTrack(-1);
```

截图

Flutter播放器SDK提供了对当前视频截图的功能，由 `setOnSnapShot` 接口实现。

```
//截图成功监听
fAliplayer.setOnSnapShot((path,playerId) {
});
//截图，path为图片保存路径
fAliplayer.snapshot(path);
```

试看

Flutter播放器SDK通过配合点播服务配置，可以实现试看功能，VidAuth播放（推荐）和VidSts播放方式支持试看功能。如何配置和使用试看功能，请参见点播试看。

配置试看功能之后，通过 `setVidAuth` 接口的 `previewTime` 方法设置播放器的试看时长。示例如下：

```
//previewTime为试看时间，单位：秒
//VidAuth播放方式
fAliplayer.setVidAuth(
        vid: "填写资源的vid",
        region: "填写资源的region",
        playAuth: "填写资源的region",
        previewTime: "填写试看时间，单位：秒");
 //VidSts播放方式
 fAliplayer.setVidSts(
        vid: "填写资源的vid",
        region: "填写资源的region",
        accessKeyId: "填写资源的鉴权id",
                accessKeySecret: "填写资源的鉴权密钥",
                securityToken: "填写资源的安全Token",
        previewTime: "填写试看时间，单位：秒");
```

其他配置

Flutter播放器SDK的其他播放配置。需要在 `prepare()` 方法之前设置给播放器。

```
var configMap = {
  'mStartBufferDuration':_mStartBufferDurationController.text,//起播缓冲区时长，单位：毫秒
  'mHighBufferDuration':_mHighBufferDurationController.text,//高缓冲时长，单位：毫秒
  'mMaxBufferDuration':_mMaxBufferDurationController.text,//最大缓冲区时长，单位：毫秒
  'mMaxDelayTime': _mMaxDelayTimeController.text,//直播最大延迟，单位：毫秒。注意：仅直播有效
  'mNetworkTimeout': _mNetworkTimeoutController.text,//网络超时时间，单位：毫秒
  'mNetworkRetryCount':_mNetworkRetryCountController.text,///网络超时重试次数，单位：毫秒
  'mMaxProbeSize': _mMaxProbeSizeController.text,//最大probe大小
  'mReferrer': _mReferrerController.text,//设置referrer
  'mHttpProxy': _mHttpProxyController.text,//http代理
  'mEnableSEI': mEnableSEI,//是否启用SEI
  'mClearFrameWhenStop': !mShowFrameWhenStop,///停止后是否清空画面
  'mDisableVideo': mDisableVideo,//禁用Video
  'mDisableAudio': mDisableAudio,//禁用Audio
  'mUserAgent':mUserAgent,//设置UserAgent
};
//应用配置
fAliplayer.setConfig(configMap);
```

性能

本地缓存

Flutter播放器SDK提供了本地缓存（边播边缓存）的功能，能够让用户重复播放视频时，达到省流量的目的。需要在 `prepare` 之前给播放器配置 `setCacheConfig` 即可实现此功能。示例如下：

```
var map = {
  "mMaxSizeMB": _mMaxSizeMBController.text,///缓存目录的最大占用空间
  "mMaxDurationS": _mMaxDurationSController.text,///设置能够缓存的单个文件的最大时长
  "mDir": _mDirController.text,///缓存目录
  "mEnable": mEnableCacheConfig,///是否开启缓存功能
};
fAliplayer.setCacheConfig(map);
```

视频下载

Flutter播放器SDK提供了点播服务视频的下载功能，能够通过VidAuth方式和VidSts方式下载点播服务上的视频。同时，下载的方式提供了安全下载和普通下载的模式（可登录点播控制台，选择**配置管理** > **分发加速配置** > **下载设置设置**）。

- **普通下载**
  下载后的视频数据未经过阿里云加密，用户可以用第三方播放器播放。
- **安全下载**
  下载后的视频数据经过阿里云加密。第三方播放器无法播放，仅支持使用阿里云的播放器SDK进行播放。

1. **创建并设置下载器。**

   ```
   FlutterAliDownloader donwloader = FlutterAliDownloader.init();
   ///设置保存路径
   donwloader.setSaveDir(path)
   ```

   下载SDK支持私有加密的下载。为了保证安全性，需要配置一个加密校验文件到SDK。

   ```
   FlutterAliPlayerFactory.initService(byteData);
   ```

2. **开始下载。**
   调用了开始下载后，会自动设置监听，并将回调信息返回。示例如下：

```
///1.prepare
  ///参数说明：type可选值为FlutterAvpdef.DOWNLOADTYPE_STS /
FlutterAvpdef.DOWNLOADTYPE_AUTH 。当type为DOWNLOADTYPE_STS时候，必填参数为：
{vid,accessKeyId,accessKeySecret,securityToken}，当 type 为 DOWNLOADTYPE_AUTH 时，必须
填参数为 {vid,playAuth}
  downloader.prepare(type, vid).then((value) {
      //value 为 map，对应 Demo 中的 DownloadModel 自定义下载类
      DownloadModel downloadModel = DownloadModel.fromJson(value);
      //2.selectItem，根据不同的 trackInfo 来确定需要下载哪个清晰度
      List<TrackInfoModel> trackInfos = downloadModel.trackInfos;
      downloader.selectItem(vid,trackInfos[0].index);
      //3.start
      downloader.start(vid, trackInfos[0].index).listen((event) {
        //说明：event 可能会有多种信息，可参考 FlutterAvpdef.EventChanneldef 中的信息，以下
为具体说明：
        if (event[EventChanneldef.TYPE_KEY] == EventChanneldef.DOWNLOAD_PROGRESS){
            //下载进度百分比信息，获取下载进度百分比：
event[EventChanneldef.DOWNLOAD_PROGRESS]
        }else if(event[EventChanneldef.TYPE_KEY] ==
EventChanneldef.DOWNLOAD_PROCESS){
            //处理进度百分比信息，获取处理进度百分比：
event[EventChanneldef.DOWNLOAD_PROCESS]
        }else if(event[EventChanneldef.TYPE_KEY] ==
EventChanneldef.DOWNLOAD_COMPLETION){
            //下载完成，可以通过 event['vid']、event['index'] 获取对应的 vid 和 index 用于
判断是哪个视频下载完成，event['savePath'] 用于获取下载完成视频的本地路径
        }else if(event[EventChanneldef.TYPE_KEY] == EventChanneldef.DOWNLOAD_ERROR)
{
            //下载失败，可以通过 event['vid']、event['index'] 获取对应的 vid 和 index 用于
判断是哪个视频下载失败，event['errorCode']、event['errorMsg'] 可以获取对应的错误码，和错误信
息
        }
      });
  });
```

3. 停止下载。

```
downloader.stop(vid, index)
```

4. 删除下载。
   删除下载选项，如果删除成功，则下载的本地文件也会随之删除。示例如下：

```
downloader.delete(vid, index)
```

5. 释放下载对象。
   当某个下载对象不再使用时，使用 `release` 方法将其释放，防止内存泄漏。示例如下：

```
downloader.release(vid, index)
```

视频加密播放

点播视频支持HLS标准加密、阿里云私有加密和DRM加密。加密播放请参见 视频加密播放。

8.Windows播放器

8.1. 快速集成

本文提供快速集成Windows播放器SDK的指引。

前提条件

- 环境中已安装Visual Studio，推荐使用Visual Studio 2017，下载地址请参见 vs_Community安装包。
- 环境中已安装QT，推荐使用QT 5.12.9，下载地址请参见 QT 5.12.9安装包。
- 已下载Windows播放器SDK，推荐下载使用最新版本。下载地址请参见 SDK简介与下载。

操作步骤

1. 解压Windows播放器SDK包。

   解压后SDK包整体结构如下所示：

目录	说明
sdk	SDK文件。
doc	帮助文档，包含JavaScript和HTML格式的文件。
demo	SDK Demo。

2. 编译Demo。
   i. 在*demo/src/AliyunPlayerTest/windows.cmake*中，修改 `set(QTDIR C:/Qt/Qt5.12.9/5.12.9/msvc2017)` 为您的QT路径。
   ii. 运行*demo/src/AliyunPlayerTest/build_win.bat*。
   iii. 双击打开*demo/src/AliyunPlayerTest/build*文件夹中生成的*QAliyunPlayerTest.sln*文件，并进行编译。
   iv. 复制*demo/src/AliyunPlayerTest/AliyunPlayerTest/mui*文件夹到您的程序目录中。
3. 集成SDK。

   复制*sdk*文件夹下的*sdk_headers*文件夹和*bin*文件夹到工程目录下，并在工程属性中设置依赖目录。此处以复制到工程的*third_party\aliplayer*目录下举例说明：
   i. 复制*sdk\sdk_headers*文件夹到*third_party\aliplayer*目录下。
   ii. 复制*sdk\bin\AliPlayer.lib*文件到*third_party\aliplayer\lib*目录下。
   iii. 在项目属性中设置项目附加包，目录为 *third_party\aliplayer\sdk_headers*。
   iv. 编译工程。

   编译成功后，复制*sdk\bin*文件夹下的所有文件夹和.dll文件到.exe文件同级目录下。

8.2. 常用功能

本文提供Windows播放器SDK常用功能的使用示例。

基础播放

创建播放器

本节介绍如何用最简单的方式让Windows播放器SDK播放视频，按照播放方式的不同可以分为手动播放和自动播放。

1. 创建播放器。

   创建AliPlayer播放器。

   ```
   using namespace alivc_player;
   AliPlayer* mPlayerPtr = AliPlayer::CreatePlayer();
   ```

2. 设置监听器。

   Windows播放器SDK提供了各种事件回调，比如：onPlayerEvent、onError等事件。使用时需要一个类继承IAVPListener，并实现里面的纯虚函数。示例如下：

   ```
   mPlayerPtr->setListener(new AVPListenerImpl);
   ```

3. 创建播放源。

   - Windows播放器SDK支持4种点播播放方式，包括：UrlSource播放、VidAuth播放（视频点播用户推荐使用）、VidSts播放、VidMps播放。
   - Windows播放器SDK仅支持1种直播播放方式，为UrlSource播放。

   点播视频播放

   点播UrlSource播放 | VidAuth播放 | 点播VidSts播放 | VidMps播放

   使用点播UrlSource播放方式播放点播视频，需要将播放器的source属性设置为播放地址。播放地址可以是第三方点播地址或阿里云点播服务中的播放地址。阿里云播放地址可以调用获取音视频播放地址接口获取。建议您集成点播服务端SDK来获取音视频播放地址，免去自签名的麻烦。调用接口获取音视频播放地址的示例请参见开发者门户 。

   点播UrlSource播放示例

   ```
   AVPUrlSource urlSource;
   urlSource.setUrl("media url");  //播放地址，可以是第三方点播地址，或阿里云点播服务中的播放地址。
   mPlayerPtr->setSource(urlSource);
   ```

   直播视频UrlSource播放

   使用UrlSource播放方式播放直播视频，需要将播放器的setUrl属性设置为直播拉流地址。播放地址可以是第三方直播地址或阿里云直播服务中的拉流地址。阿里云直播拉流地址可以通过直播控制台的地址生成器生成。详情请参见阿里云直播地址生成器 。

   ```
   AVPUrlSource urlSource;
   urlSource.setUrl("media url");  //播放地址可以是第三方直播地址，或阿里云直播服务中的拉流地址。
   mPlayerPtr->setSource(urlSource);
   ```

4. 设置显示View。

   如果播放源有画面，那么需要设置显示的view到播放器中，用来显示画面。示例如下：

   ```
   mPlayerPtr->setView((void *)显示区域的HWND);
   ```

5. 准备播放。

   调用 `prepare()` 方法准备播放。

   ```
   mPlayerPtr->prepare();
   ```

6. （可选）开启自动播放，默认为关闭状态。

   ```
   mPlayerPtr->setAutoPlay(true);
   ```

7. 开始播放。

- 如果未开启自动播放，需要在 `OnPrepard` 回调中调用 `mPlayerPtr->start();` 开始播放视频。
- 如果开启了自动播放，则不需要调用 `mPlayerPtr->start();` ，数据解析完成后将开始自动播放视频。

```
mPlayerPtr->start();
```

控制播放

Windows播放器SDK支持开始、暂停、从指定时间点播放等主流操作。 指定时间播放即seek。

开始播放

开始播放视频，由 `start` 接口实现。示例如下：

```
mPlayerPtr->start();
```

从指定时间开始播放

跳转到某个时刻进行播放，由 `seekToTime` 接口实现。适用于用户拖拽进度条，或续播等需要从指定时间点开始播放的场景。示例如下：

```
//跳转到特定时刻的视频画面。
//mode取值：AVP_SEEKMODE_ACCURATE（精确seek）和AVP_SEEKMODE_INACCURATE（非精确seek）。非精确seek会跳转到向前最近的关键帧处。精确seek会跳转到精确的时间，但比非精确seek会慢一些。
mPlayerPtr->seekToTime(int64_t time_in_ms, mode);
```

暂停播放

暂停播放视频，由 `pause` 接口实现。示例如下：

```
mPlayerPtr->pause();
```

停止播放

停止播放视频，由 `stop` 接口实现。示例如下：

```
mPlayerPtr->stop();
```

设置显示模式

Windows播放器SDK支持填充、旋转、镜像等显示设置。

填充

支持设置宽高比适应、宽高比填充和拉伸填充这3种画面填充模式，由 `setScalingMode` 接口实现。示例如下：

```
//设置宽高比适应（将按照视频宽高比等比缩小到view内部，不会有画面变形）
mPlayerPtr->setScalingMode(AVP_SCALINGMODE_SCALEASPECTFIT);
//设置宽高比填充（将按照视频宽高比等比放大，充满view，不会有画面变形）
mPlayerPtr->setScalingMode(AVP_SCALINGMODE_SCALEASPECTFILL);
//设置拉伸填充（如果视频宽高比例与view比例不一致，会导致画面变形）
mPlayerPtr->setScalingMode(AVP_SCALINGMODE_SCALETOFILL);
```

旋转

画面按指定角度旋转，由 `setRotateMode` 接口实现。设置后还可查询旋转角度。示例如下：

```
//设置画面顺时针旋转0度
mPlayerPtr->setRotateMode(AVP_ROTATE_0);
//设置画面顺时针旋转90度
mPlayerPtr->setRotateMode(AVP_ROTATE_90);
//设置画面顺时针旋转180度
mPlayerPtr->setRotateMode(AVP_ROTATE_180);
//设置画面顺时针旋转270度
mPlayerPtr->setRotateMode(AVP_ROTATE_270);
//获取旋转角度
mPlayerPtr->getRotateMode();
```

镜像

支持水平镜像、垂直镜像和无镜像，由 `setMirrorMode` 接口实现。示例如下：

```
//设置无镜像
mPlayerPtr->setMirrorMode(AVP_MIRRORMODE_NONE);
//设置水平镜像
mPlayerPtr->setMirrorMode(AVP_MIRRORMODE_HORIZONTAL);
//设置垂直镜像
mPlayerPtr->setMirrorMode(AVP_MIRRORMODE_VERTICAL);
```

获取播放信息

Windows播放器SDK支持获取当前的播放进度和播放时长。

获取当前播放进度

获取当前的播放时刻，在 `onCurrentPositionUpdate` 回调中获取，单位毫秒。示例如下：

```
void AlivcLivePlayerMainDlg::onCurrentPositionUpdate(AliPlayer *player, int64_t positio
n)
{
    //position为当前播放进度，单位为毫秒
printf("current position is %lld ms", position);
}
```

获取播放时长

获取视频总时长。需要在视频加载完成以后才可以获取到，可以在 `AVPEventPrepareDone` 事件后获取 `duration` 。示例如下：

```
void AlivcLivePlayerMainDlg::onPlayerEvent(AliPlayer *player, AVPEventType eventType)
{
    if (eventType == AVPEventPrepareDone) {
        int64_t duration = mPlayerPtr->getDuration();
printf("total duration is %lld ms", duration);
    }
}
```

获取缓冲进度

获取视频当前的缓冲进度，需要在 `onBufferedPositionUpdate` 回调中获取 `position` 。示例如下：

```
void AlivcLivePlayerMainDlg::onBufferedPositionUpdate(AliPlayer *player, int64_t positi
on)
{
    printf("buffered position is %lld ms", position);
}
```

监听播放状态

监听播放器的状态，`onPlayerStatusChanged`回调参数为当前播放器状态。示例如下：

```
void AlivcLivePlayerMainDlg::onPlayerStatusChanged(AliPlayer *player, AVPStatus oldStat
us, AVPStatus newStatus)
{
    switch (newStatus) {
case AVPStatusIdle:{
//  空转，闲时，静态
}
break;
case AVPStatusInitialzed:{
//  初始化完成
}
 break;
     case AVPStatusPrepared:{
//  准备完成
}
break;
case AVPStatusStarted:{
//  正在播放
}
 break;
case AVPStatusPaused:{
//  播放暂停
}
break;
case AVPStatusStopped:{
//  播放停止
 }
break;
case AVPStatusCompletion:{
//  播放完成
 }
break;
case AVPStatusError:{
//  播放错误
 }
break;
default:
break;
}
}
```

设置音量

设置音量包括音量调节和静音设置。

音量调节

调节音量大小，由 `setVolume` 接口实现。设置后还可获取音量信息。示例如下：

```
//volume的值为0～2.0。
mPlayerPtr->setVolume(1.0f);
//获取音量信息。
mPlayerPtr->getVolume();
```

静音设置

将播放中的视频设置为静音状态，由 `setMute` 接口实现。示例如下：

```
mPlayerPtr->setMute(true);
```

倍速播放

Windows播放器SDK提供了倍速播放视频的功能，通过设置 `rate` 方法，能够以0.5倍~2倍速去播放视频。同时保持变声不变调。示例如下：2022.03.18：依然仅支持0.5倍~2倍，暂不支持5倍

```
//设置倍速播放：支持0.5～2倍速的播放，通常按0.5的倍数来设置，例如0.5倍、1倍、1.5倍等
mPlayerPtr->setRate(1.5);
```

多清晰度设置

如果使用VID方式（VidAuth（推荐）及VidSts）播放，无需额外设置。Windows播放器SDK会从视频点播服务获取清晰度列表。Windows播放器SDK支持获取和切换清晰度，UrlSource播放方式暂不支持此设置。

获取清晰度

当视频加载完成后，可以获取视频的清晰度。

```
void AlivcLivePlayerMainDlg::onTrackReady(AliPlayer *player, AVPTrackInfo *info[], int
count)
{
    if (count < 0) {
        return;
    }
    for (int i = 0; i < count; i++) {
        AVPTrackInfo *track = info[i];
        switch (track->trackType) {
            case AVPTRACK_TYPE_VIDEO: {
                int trackBitrate = track->trackBitrate;
            } break;
        }
    }
}
```

切换清晰度

通过 `selectTrack` 方法切换清晰度，传递对应TrackInfo的index即可。

```
mPlayerPtr->selectTrack(trackIndex);
```

清晰度切换通知

清晰度切换成功回调。

```
void AlivcLivePlayerMainDlg::onTrackChanged(AliPlayer *player, AVPTrackInfo *info)
{
    // 切换完成
}
```

循环播放

Windows播放器SDK提供了循环播放视频的功能。调用 `setLoop` 开启循环播放，播放完成后，将会自动从头开始播放视频。同时循环开始的回调将会在 `AVPEventLoopingStart` 中通知。示例如下：

```
mPlayerPtr->setLoop(true);
```

进阶播放

外挂字幕

Windows播放器SDK支持添加和切换外挂字幕，现仅支持SRT格式。示例如下：

1. 创建显示字幕的控件。

   根据不同的字幕格式创建不同的View。

   ```
   mSubtitleLabelPtr = new QLabel(QString(""), this);
   mSubtitleLabelPtr->setParent(this);
   ```

2. 设置字幕相关监听。

   ```
   //外挂字幕被添加
   void AlivcLivePlayerMainDlg::onSubtitleExtAdded(AliPlayer *player, int64_t trackInd
   ex, const char *URL)
   {
       emit SgnAddExtSubtitle((qint64) trackIndex);
   }
   //字幕显示回调
   void AlivcLivePlayerMainDlg::onSubtitleShow(AliPlayer *player, int64_t trackIndex,
   int64_t subtitleId, const char *subtitle)
   {
       emit SgnUpdateSubtitle(QString::fromStdString(subtitle), true);
   }
   //字幕隐藏回调
   void AlivcLivePlayerMainDlg::onSubtitleHide(AliPlayer *player, int64_t trackIndex,
   int64_t subtitleId)
   {
       emit SgnUpdateSubtitle(QString(""), false);
   }
   ```

3. 添加字幕。

   ```
   mPlayerPtr->addExtSubtitle("url");
   ```

4. 切换字幕。

   ```
   mPlayerPtr->selectExtSubtitle(trackIndex, true);
   ```

纯音频播放

通过禁用视频播放，达到纯音频播放的效果。在prepare之前配置PlayerConfig。

```
AVPConfig *config = mPlayerPtr->getConfig();
config->mDisableVideo = true;
mPlayerPtr->setConfig(config);
```

2022.03.18宇源确认Windows没有提供硬解码能力，将该功能屏蔽掉。

网络自适应切换视频清晰度

Windows播放器SDK支持多码率自适应HLS、DASH视频流。在 `prepare` 成功之后，通过 `getMediaInfo` 可以获取到各个码流的信息，即 `TrackInfo` 。示例如下：

```
AVPMediaInfo info = mPlayerPtr->getMediaInfo();
for (int i = 0; i < info.trackCount; i++) {
     AVPTrackInfo *track = info.tracks[i];
}
```

> **说明** HLS、DASH视频流需要经过多码率视频转码模板组打包处理，可以在点播控制台的 **配置管理** > **媒体处理配置** > **转码模板组** 中配置生产对应的视频流，详细操作请参见 **视频或字幕打包模板设置** 。

在播放过程中，可以通过调用播放器的 `selectTrack` 方法切换播放的码流，取值为SELECT_AVPTRACK_TYPE_VIDEO_AUTO时，为多码率自适应。示例如下：

```
//多码率切换
mPlayerPtr->selectTrack(trackIndex);
//切换为自适应码率
mPlayerPtr->selectTrack(SELECT_AVPTRACK_TYPE_VIDEO_AUTO);
```

切换的结果会在 `onTrackChanged` 监听之后会回调。示例如下：

```
void AlivcLivePlayerMainDlg::onTrackChanged(AliPlayer *player, AVPTrackInfo *info)
{
    if (info->trackType == AVPTRACK_TYPE_VIDEO) {
        // video changed
    }
    // etc
}
```

截图

Windows播放器SDK提供了对当前视频截图的功能，由 `snapShot` 接口实现。截取的是原始的argb32格式的数据，回调接口为 `onSnapshotImageBuffer` 。示例如下：

```
//截取当前播放的画面
mPlayerPtr->snapshot();
//截图回调
void AlivcLivePlayerMainDlg::onSnapshotImageBuffer(AliPlayer *player, int width, int he
ight, unsigned char *pARGBBuffer)
{
    QString savePath = "save path";
    QImage snapshot(pARGBBuffer, width, height, QImage::Format_ARGB32);
    snapshot.save(savePath);
}
```

> 注意 截图不包含界面。

试看

Windows播放器SDK通过配合点播服务配置，可以实现试看功能，VidAuth播放（推荐）和VidSts播放方式支持试看功能。如何配置和使用试看功能，请参见点播试看。

配置试看功能之后，通过 `VidPlayerConfigGenerator` 接口的 `setPreviewTime` 方法设置播放器的试看时长。当设置试看的时长，通过播放器SDK播放视频时，服务端将不会返回完整的视频内容，而是返回试看时间段的内容。以VidSts播放方式为例，示例如下：

```
VidPlayerConfigGenerator playConfigGen;
playConfigGen.setPreviewTime(10);
AVPVidStsSource vidSource;
vidSource.initWithVid(视频vid,
                      "accessKeyId",
                      "accessKeySecret",
                      "安全token",
                      "接入地域",
                      playConfigGen.generatePlayerConfig());
mPlayerPtr->setSource(vidSource);
```

设置Referer

Windows播放器SDK支持设置Referer，配合控制台的黑白名单Referer，可以控制访问权限，由 `AVPConfig` 方法设置请求Referer。示例如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//设置referer
pConfig->referer = referer;
....//其他设置
//给播放器设置配置
mPlayerPtr->setConfig(pConfig);
```

设置UserAgent

Windows播放器SDK提供了AVPConfig用来设置请求UA。设置之后，播放器请求的过程中将会带上UA信息。示例如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//设置userAgent
pConfig->userAgent = userAgent;
....//其他设置
//设置配置给播放器
mPlayerPtr->setConfig(pConfig);
```

配置网络重试时间和次数

Windows播放器SDK支持设置网络超时的时间和重试次数，由 `AVPConfig` 接口实现。示例如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//设置网络超时时间，单位ms
pConfig->networkTimeout = 5000;
//设置超时重试次数。每次重试间隔为networkTimeout。networkRetryCount=0则表示不重试，重试策略App决定，默认值为2
pConfig->networkRetryCount = 2;
....//其他设置
//给播放器设置配置
mPlayerPtr->setConfig(pConfig);
```

> **说明**
>
> - 如果设置了networkRetryCount，若此时发生网络问题，导致出现loading后，那么将会重试networkRetryCount次，每次的间隔时间为networkTimeout。
> - 如果重试次数超过上限值之后，还是loading的状态，那么就会回调onError事件，此时AVPErrorModel.code为ERROR_LOADING_TIMEOUT。
> - 如果networkRetryCount设置为0，当网络重试超时的时候，播放器就会回调onPlayerEvent，参数eventWithString为EVENT_PLAYER_NETWORK_RETRY。此时可以调用播放器的reload方法进行重新加载网络，或者进行其他的处理。

配置缓存和延迟控制

对于播放器来说，缓存的控制非常重要。合理的配置可以有效的加快起播速度并减少卡顿。Windows播放器SDK通过 `AVPConfig` 提供了设置缓存和延迟的控制接口。示例如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//最大延迟。注意：仅直播有效。当延时比较大时，播放器SDK内部会追帧等，保证播放器的延时在这个范围内。
pConfig->maxDelayTime = 5000;
//最大缓冲区时长。单位ms。播放器每次最多加载所设置时长的缓冲数据。
pConfig->maxBufferDuration = 50000;
//高缓冲时长。单位ms。当网络不好导致加载数据时，如果加载的缓冲时长到达这个值，结束加载状态。
pConfig->highBufferDuration = 3000;
// 起播缓冲区时长。单位ms。时间设置越短，起播越快。也可能会导致播放之后很快就会进入加载状态。
pConfig->startBufferDuration = 500;
//其他设置
//给播放器设置配置
mPlayerPtr->setConfig(pConfig);
```

> 注意 三个缓冲区时长的大小关系必须为：startBufferDuration ≤ highBufferDuration ≤ maxBufferDuration。

设置HTTP Header

播放器通过 `AVPConfig` 参数，可以给播放器中的请求加上HTTP的header参数。示例如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//设置header
pConfig->headerCount = 1;
pConfig->httpHeaders = new char *[pConfig->headerCount];
//比如使用httpdns时，需要设置Host。
pConfig->httpHeaders[0] = strdup("Host:example.com");
....//其他设置
//设置配置给播放器
mPlayerPtr->setConfig(pConfig);
```

性能

本地缓存

Windows播放器SDK提供了本地缓存（边播边缓存）的功能，能够让用户重复播放视频时，达到省流量的目的。只需在prepare之前给播放器配置AVPCacheConfig即可实现此功能。示例如下：

```
AVPCacheConfig mCacheConfig;
//开启缓存功能
mCacheConfig.enable = true;
//能够缓存的单个文件最大时长。超过此长度则不缓存
mCacheConfig.maxDuration = 100;
//缓存目录的最大大小。超过此大小，将会删除最旧的缓存文件
mCacheConfig.maxSizeMB = 200;
//缓存目录的位置，需替换成APP期望的路径
mCacheConfig.path = strdup("please use your cache path here");
//设置缓存配置给到播放器
mPlayerPtr->setCacheConfig(&mCacheConfig);
```

缓存成功之后，以下情况将会利用缓存文件（必须已经设置了setCacheConfig）：

- 如果设置了循环播放，那么第二次播放的时候，将会自动播放缓存的文件。
- 缓存成功后，重新创建播放器，播放同样的资源，也会自动使用缓存文件。

> 注意 VID的缓存文件是通过vid等信息定位的，所以VID的缓存文件将需要网络请求信息之后才能确定使用哪个缓存文件。

边播边缓存的使用有限制条件，具体如下：

- 对于UrlSource播放方式。如果是HLS（即M3U8）地址，将不会缓存。如果是其他支持的格式，则根据缓存配置进行缓存。
- 对于VidAuth、VidSts播放方式，将会根据缓存配置进行缓存。
- 播放器读取完全部的数据则视为缓存成功。如果在此之前，调用 `stop` ，或者出错 `onError` ，则缓存将会失败。
- cache内的seek的操作不会影响缓存结果。cache外的seek会导致缓存失败。
- 如果加密文件与APP信息不一致，将会缓存失败。

- cache的结果回调，会通过 `onPlayerEvent` 回调，包括 `EVENT_PLAYER_CACHE_SUCCESS` 和 `EVENT_PLAYER_CACHE_ERROR` 。

视频下载

Windows播放器SDK提供了点播服务视频的下载功能，能够通过VidAuth方式和VidSts方式下载点播服务上的视频。同时，下载的方式提供了安全下载和普通下载的模式（可登录点播控制台，选择配置管理 > 分发加速配置 > 下载设置设置）。

- 普通下载：使用点播服务加密视频后，下载的视频数据也不是经过阿里云加密的。用户可以用第三方播放器播放。
- 安全下载：使用点播服务没有加密视频，下载后的视频数据也是经过阿里云加密的。用户用第三方播放器是播放不了的。只能使用阿里云的播放器去播放。

1. 创建下载器。示例如下：

```
mMediaDownloader = AliMediaDownloader::CreateMediaDownloader();
mMediaDownloader->setListener(new AVDListenerImpl);
mMediaDownloader->setSaveDirectory("saveDir");
```

下载SDK支持私有加密的下载。为了保证安全性，还需要设置一个加密校验信息，传入encryptedApp.dat文件的内容。示例如下：

```
InitPrivateService(fileContentBuffer, fileLength);
```

2. 准备下载源。
通过prepare方法准备下载源，仅支持VidAuth（推荐）和VidSts方式的下载源。以VidSts方式的下载源为例，示例如下：

```
AVPVidStsSource vidSource;
vidSource.initWithVid(视频vid,
                      "accessKeyId",
                      "accessKeySecret",
                      "安全token",
                      "接入地域",
                      nullptr);
mMediaDownloader->prepareWithVid(&vidSource);
```

3. 准备成功后选择下载项。
准备成功后，会回调 `onPrepared` 方法。选择某个Track进行下载，示例如下：

```
void YourClass::onPrepared(AliMediaDownloader *downloader, AVPMediaInfo *mediaInfo)
{
    AVPTrackInfo *track = mediaInfo->tracks[0];
    mMediaDownloader->selectTrack(track->trackIndex);
}
```

4. 更新下载源并开始下载。
完成以上操作后，开始下载（为了防止VidSts和VidAuth过期，最好先更新下载源的信息）。示例如下：

```
mMediaDownloader->updateWithVid(&vidSource);
mMediaDownloader->start();
```

5. 下载成功或失败后，释放下载器。
下载成功后，释放下载器。示例如下：

```
delete mMediaDownloader;
mMediaDownloader = nullptr;
```

相关文档
接口说明

8.3. 接口说明

本文为您提供了Windows播放器SDK的接口文档。

SDK版本	链接
v5.2.0	Windows播放器SDK接口文档

8.4. RTS播放

您可以阅读本文，了解播放器集成RTS SDK播放RTS的操作步骤。

集成RTS SDK

下面为您介绍集成RTS SDK播放RTS流的操作步骤：

1. 集成Windows播放器。

   集成Windows播放器，具体操作，请参见集成Windows播放器。

   > 说明 Windows播放器SDK自带RTS SDK（RtsSDK.dll 、cicada_plugin_artcSource.dll），不需要额外下载配置。

2. 设置播放器最大缓冲延迟等。

   播放器SDK通过AVPConfig提供了MaxDelayTime等参数设置播放直播流最大延迟缓存的接口。RTS场景下，建议的参数设置值如下：

```
//先获取配置
AVPConfig *pConfig = mPlayerPtr->getConfig();
//设置最大延迟为1000，延迟控制交由RTS控制
pConfig->maxDelayTime = 1000;
//设置播放器启播缓存为10ms，数据控制由RTS控制。
pConfig->highBufferDuration = 10;
pConfig->startBufferDuration = 10;
//其他设置
//...
//设置配置给播放器
mPlayerPtr->setConfig(pConfig);
```

9.最佳实践

9.1. 视频加密播放

本文为您提供了SDK加密播放的操作步骤及说明，通过本文您可以了解如何进行视频SDK加密以及相关设置。

基本介绍

阿里云视频点播已经有一套完善的机制保障视频内容的安全、不被盗链、非法下载和传播，安全机制如下表：

安全机制	安全手段	特点	安全等级	使用门栏
访问限制	Referer、UA黑白名单	基于HTTP Header跟踪来源，但极易伪造。	低	低，仅云端配置。
	IP黑白名单	拒绝或只允许特定IP访问、不适合大量C端用户的分发。	较低	低，仅云端配置。
	URL访问次数、独立IP数限制	设置访问上限，不适合大量分发也无法识别合法访问。	较低	低，仅云端配置。
播放中心鉴权	URL鉴权、URL时效控制	生成动态变化的加密URL，可自定义过期时效。	中	较低，通过API获取动态地址，或根据Key自动生成。
业务方二次鉴权	透传业务请求信息给客户自定义鉴权中心来判断合法性	业务方更懂自己的用户：客户添加自定义的业务请求信息，通过自建鉴权中心，更加精准识别合法请求。	较高	较高，需部署鉴权中心，并确保高可用。
视频加密	阿里云视频加密（私有加密）	云端一体的视频加密解决方案，采用私有加密算法，确保链路的安全传输。	高	较低，简单配置+集成阿里云播放器即可。
	HLS标准加密	HLS标准加密方案，使用AES-128进行内容加密，适配所有HLS播放器，但密钥易被窃取。	较高	高，需自建密钥管理令牌颁发服务，并确保链路传输安全。
	商业DRM	支持多种DRM技术： • 苹果Fairplay • 谷歌Widevine 各平台原生支持确保安全性。	高	高，单独购买License，集成特定SDK。

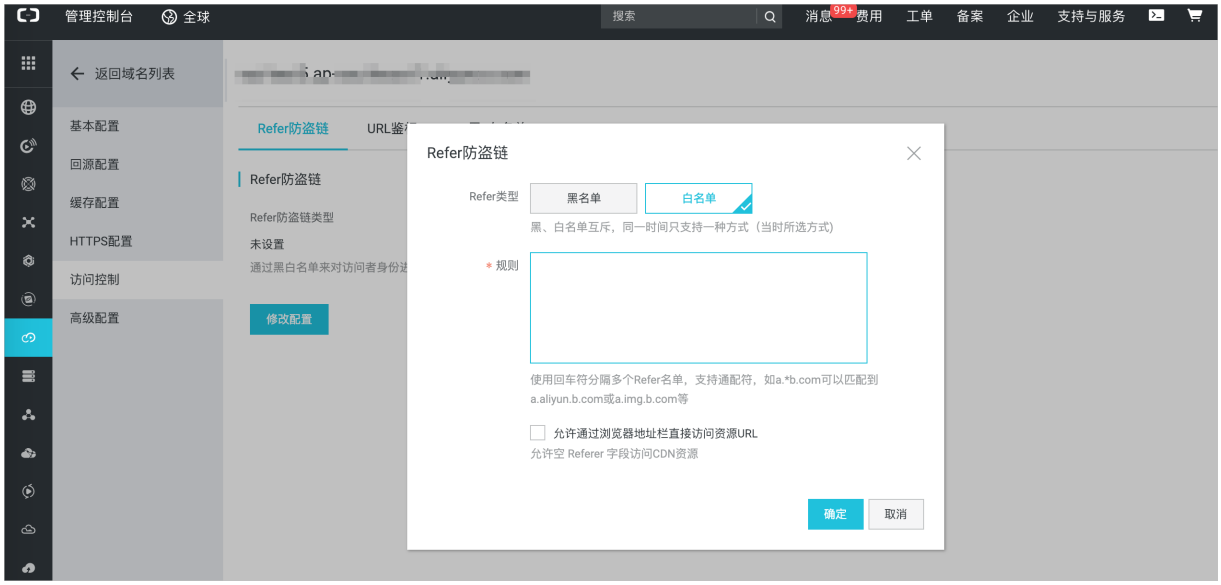
Referer访问控制

基于HTTP协议支持的Referer机制，通过Referer跟踪来源，对来源进行识别和判断，用户可配置访问的Referer黑、白名单（二者互斥）来限制视频资源被访问的情况。

- 阿里云控制台支持黑名单和白名单两种模式，访客对资源发起请求后，请求到达CDN节点，节点会根据用户预设的防盗链黑名单或白名单进行过滤，符合规则可顺利请求到视频数据；若不符合，请求会被拒绝，并返回403响应码。
- 配置后会自动添加泛域名支持，例如填写 `example.com` ，最终配置生效的是 `*.example.com` ，所有子级域名都会生效。

- 由于移动端一般获取不到Referer，当前默认支持空Referer访问，可选择关闭。

控制台配置页面如下：



播放器referer接口

阿里云播放器提供了Referer的设置。比如：控制台设置的白名单为 `aliyundoc.com` 。

- Android端，示例如下：

```
AliPlayer player = AliPlayerFactory.createAliPlayer(context);
....
//获取播放器参数
PlayerConfig config = player.getConfig();
//设置referrer。注意：加上http(s)://的协议头
config.mReferrer = "http://aliyundoc.com";
//设置回播放器
player.setConfig(config);
```

- iOS端，示例如下：

```
//获取播放器参数
AVPConfig* config = [self.player getConfig];
//设置referrer。注意：加上http(s)://的协议头
config.referer = @"http://aliyundoc.com";
//设置回播放器
[self.player setConfig:config];
[self.player setStsSource:self.stsSource];
[self.player prepare];
```

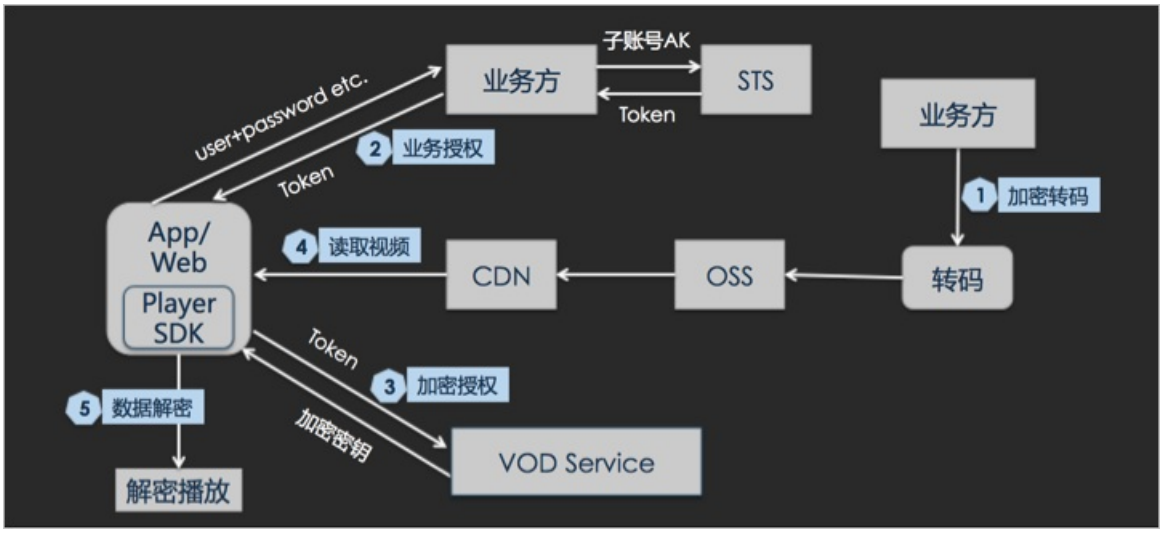
视频加密播放

防盗链安全机制能有效保障用户的合法访问，但对于付费观看视频的场景，用户只需通过一次付费行为拿到视频合法的防盗链播放URL，将视频下载到本地，进而实现二次分发。因此，防盗链方案对于视频版权保护是远远不够的。视频文件一旦泄露，会给付费观看模式造成十分严重的经济损失。

阿里云视频加密是对视频数据加密，即使下载到本地，视频本身也是被加密的，无法恶意二次分发，可有效防止视频泄露和盗链问题。

- 阿里云私有加密

阿里云视频加密采用私有的加密算法和安全传输机制，提供云端一体的视频安全方案，核心部分包括 **加密转码和解密播放**。示意图如下：



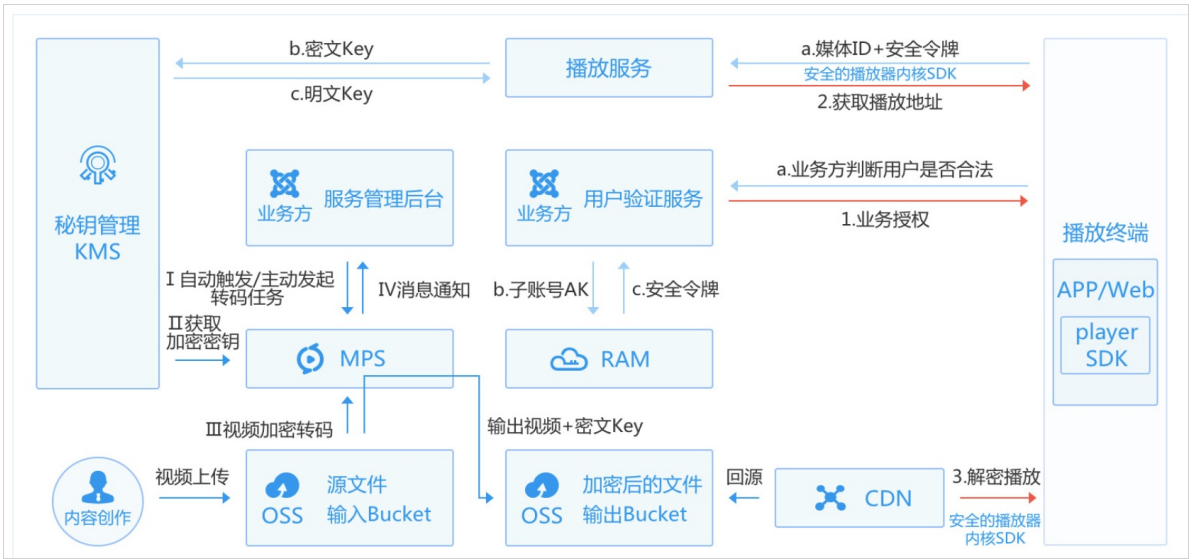
- 核心优势
  - 每个媒体文件拥有独立的加密钥匙，能有效避免采用单一密钥时，一个密钥的泄露引起大范围的安全问题。
  - 提供信封加密机制“密文Key+明文Key”，仅密文Key入库，明文Key不落存储，所有过程只在内存中，用完即销毁。
  - 提供安全的播放器内核SDK，涵盖iOS/Android/多平台，自动对加密内容进行解密播放。
  - 播放器和云端使用私有加密协议进行密文传输，不传输明文Key，有效防止密钥被窃取。
  - 提供安全下载，缓存到本地的视频会再次加密，在确保无网离线播放前提下，防止视频被拷贝窃取。

  > 注意 阿里云视频加密仅支持输出HLS格式，且只能使用阿里云播放器。更多信息请参见 **阿里云视频加密**。

- 如何播放私有加密
  阿里云播放器将内部解密逻辑、服务端交互逻辑都封装到了SDK内部。播放加密视频和普通方式没有区别，不用做多余属性的设置，只需要通过videoId的播放方式集成播放器播放视频即可。所以您只需要配置加密转码即可，您可以零成本使用加密播放。

- HLS标准加密

HLS标准加密支持HTTP Live Streaming中规定的通用加密方案，使用AES-128对视频内容本身进行加密，同时能支持所有的HLS播放器，您可选择使用自研或开源的播放器。相比私有加密方案灵活性更好，但使用门槛更高、安全性更低，流程图如下：



i. 搭建密钥管理服务，提供密钥生成（用于转码时对视频内容进行加密）和解密服务（用于播放时获取解密密钥），也可基于阿里云KMS进行封装。

ii. 提供令牌颁发服务，用于验证播放端的身份，避免解密密钥被非法获取。

iii. 播放器和云端传输明文Key，容易被窃取。更多内容，请参见 HLS标准加密 。

- 如何播放HLS标准加密
  阿里云播放器支持用户令牌传递，阿里云CDN服务会动态修改M3U8文件中的解密URI，解密URI中会带上用户的令牌，业务方可以对令牌进行验证。
- STS设置HlsUriToken
  - Android端：通过VidSts播放源来设置playConfig，playConfig中携带mtsHlsUriToken参数。示例如下：

```
VidSts vidsts = new VidSts();
VidPlayerConfigGen playerConfig = new VidPlayerConfigGen();
//设置用户令牌
playerConfig.setMtsHlsUriToken("用户令牌");
vidsts.setPlayConfig(playerConfig);
```

  - iOS端：通过AVPVidStsSource播放来设置playConfig，playConfig中携带mtsHlsUriToken参数。示例如下：

```
- (instancetype)initWithVid:(NSString *)vid
                  accessKeyId:(NSString *)accessKeyId
              accessKeySecret:(NSString *)accessKeySecret
                securityToken:(NSString *)securityToken
                       region:(NSString *)region
                   playConfig:(NSString *)playConfig;
- (void)setStsSource:(AVPVidStsSource*)source;
```

  - 生成playConfig参数
    SDK中提供了一个类：VidPlayerConfigGenerator，通过这个类的setHlsUriToken来设置令牌，然后最终通过generatePlayerConfig接口来生成playConfig。示例如下：

```
-(void) setHlsUriToken:(NSString*)MtsHlsUriToken;
-(NSString*) generatePlayerConfig;
```

PlayAuth设置HlsUriToken

- Android端：通过VidAuth播放源来设置playConfig，playConfig中携带mtsHlsUriToken参数。示例如下：

```
VidAuth vidauth = new VidAuth();
VidPlayerConfigGen playerConfig = new VidPlayerConfigGen();
//设置用户令牌
playerConfig.setMtsHlsUriToken("用户令牌");
vidauth.setPlayConfig(playerConfig);
```

- iOS端：通过AVPVidAuthSource播放来设置playConfig，playConfig中携带mtsHlsUriToken参数。示例如下：

```
- (instancetype)initWithVid:(NSString *)vid
        playAuth:(NSString *)playAuth
          region:(NSString *)region
          playConfig:(NSString *)playConfig;
- (void)setAuthSource:(AVPVidAuthSource*)source;
```

- 生成playConfig参数
  SDK中提供了一个类：VidPlayerConfigGenerator，通过这个类的setHlsUriToken来设置令牌，然后最终通过generatePlayerConfig接口来生成playConfig。示例如下：

```
-(void) setHlsUriToken:(NSString*)MtsHlsUriToken;
-(NSString*) generatePlayerConfig;
```

MPS设置HlsUriToken

- Android端：创建VidMps对象后，通过调用setHlsUriToken方法设置，然后传递给播放器播放即可。示例如下：

```
VidMps vidmps = new VidMps();
//设置用户令牌
vidmps.setHlsUriToken("用户令牌");
... // vidmps的其他设置。
AliPlayer player = AliPlayerFactory.createAliPlayer(context);
//设置vidmps到播放器
player.setDataSource(vidmps);
...//播放器的其他设置
```

- iOS端：通过AVPVidMpsSource播放来设置mtsHlsUriToken。示例如下：

```
- (instancetype)initWithVid:(NSString*)vid
                  accId:(NSString *)accId
              accSecret:(NSString*)accSecret
               stsToken:(NSString*)stsToken
               authInfo:(NSString*)authInfo
                 region:(NSString*)region
             playDomain:(NSString*)playDomain
         mtsHlsUriToken:(NSString*)mtsHlsUriToken;
- (void)setMpsSource:(AVPVidMpsSource*)source;
```

9.2. 安全下载

阿里云SDK提供了点播服务视频的下载功能，通过本文节您将了解Android端与iOS端使用SDK安全下载的操作步骤、示例及说明。

简介

阿里云SDK提供了点播服务视频的下载功能，能够通过VidSts和VidAuth下载点播服务上的视频，安全下载可确保下载的视频为加密视频，仅能通过在控制台下载加密文件时填写的bundleID或签名绑定的应用进行播放。

> **说明**
> - 需要保护视频版权的用户使用，仅限点播用户。
> - 标准加密使用播放器可以缓存到本地，需要视频点播开启安全下载功能。
> - 安全下载视频为加密视频，请参见 HLS标准加密（私有加密）。
> - 私有加密暂不支持IOS系统的网页端播放。

Android端关键实现

1. 创建并设置下载器。

   创建下载器，通过AliDownloaderFactory创建。示例如下：

   ```
   AliMediaDownloader mAliDownloader = null;
   ......
   //创建下载器
   mAliDownloader = AliDownloaderFactory.create(getApplicationContext());
   //配置下载保存的路径
   mAliDownloader.setSaveDir("保存的文件夹地址");
   ```

   如果下载的视频是经过阿里云加密转码过后的，那么还需要设置一个加密校验信息文件（encryptedApp.dat）获取方式，请参见安全文件获取问题。示例如下：

   ```
   PrivateService.initService(getApplicationContext(), "encryptedApp.dat的本地路径");
   ```

2. 设置监听事件。

   下载器提供了多个事件监听。示例如下：

```
        mAliDownloader.setOnPreparedListener(new
AliMediaDownloader.OnPreparedListener() {
            @Override
            public void onPrepared(MediaInfo mediaInfo) {
                //准备下载项成功
            }
        });
        mAliDownloader.setOnProgressListener(new
AliMediaDownloader.OnProgressListener() {
            @Override
            public void onDownloadingProgress(int percent) {
                //下载进度百分比
            }
            @Override
            public void onProcessingProgress(int percent) {
               //处理进度百分比
            }
        });
        mAliDownloader.setOnErrorListener(new AliMediaDownloader.OnErrorListener()
{
            @Override
            public void onError(ErrorInfo errorInfo) {
                //下载出错
            }
        });
        mAliDownloader.setOnCompletionListener(new
AliMediaDownloader.OnCompletionListener() {
            @Override
            public void onCompletion() {
                //下载成功
            }
        });
```

> **说明** 下载进度与处理进度的回调区别：下载进度完成前，是需要进行网络交互的；处理进度回调时，无需网络交互。

3. **准备下载源。**

   通过preapre**方法准备下载源。下载源支持**VidSts**和**VidAuth**两种。以**VidSts**举例。示例如下：**

```
//创建VidSts
 VidSts aliyunVidSts = new VidSts();
 aliyunVidSts.setVid(视频vid);
 aliyunVidSts.setAccessKeyId(临时akId);
 aliyunVidSts.setAccessKeySecret(临时akSecret);
 aliyunVidSts.setSecurityToken(安全token);
 aliyunVidSts.setRegion(接入区域);
//准备下载源
 mAliDownloader.prepare(aliyunVidSts)
```

4. **选择下载任务。**

   准备成功后，会回调 OnPreparedListener**方法。选择某个**Track**进行下载，示例如下：**

```
public void onPrepared(MediaInfo mediaInfo) {
      //准备下载项成功
      List<TrackInfo> trackInfos = mediaInfo.getTrackInfos();
     //比如：下载第一个TrackInfo
      mAliDownloader.selectItem(trackInfos.get(0).getIndex());
}
```

5. 更新下载源并开始下载。

   完成以上操作步骤后，开始下载。示例如下：

```
//更新下载源
  mAliDownloader.updateSource(mVidSts);
//开始下载
  mAliDownloader.start();
```

> 说明 为了防止VidSts和VidAuth过期，请更新下载源的信息。

6. 下载成功或失败后，释放下载器。

   下载成功后，调用release释放下载器。在onCompletion或者onError回调中。示例如下：

```
mAliDownloader.stop();
mAliDownloader.release();
```

iOS系统关键实现

1. 创建并设置下载器。

   创建下载器，示例如下：

```
AliMediaDownloader *downloader = [[AliMediaDownloader alloc] init];
[downloader setSaveDirectory:self.downLoadPath];
[downloader setDelegate:self];
```

   如果下载的视频是经过阿里云加密转码过后的，那么还需要设置一个加密校验信息文件（encryptedApp.dat）获取方式，请参见安全文件获取问题 。示例如下：

```
NSString *encrptyFilePath = [[NSBundle mainBundle] pathForResource:@"encryptedApp"
ofType:@"dat"];
[AliPrivateService initKey:encrptyFilePath];
```

2. 设置监听事件。

   下载器提供了多个事件监听。示例如下：

```
-(void)onPrepared:(AliMediaDownloader *)downloader mediaInfo:(AVPMediaInfo *)info {
    //准备下载项成功
}
-(void)onError:(AliMediaDownloader *)downloader errorModel:(AVPErrorModel
*)errorModel {
    //下载出错
}
-(void)onDownloadingProgress:(AliMediaDownloader *)downloader percentage:
(int)percent {
    //下载进度百分比
}
-(void)onProcessingProgress:(AliMediaDownloader *)downloader percentage:
(int)percent {
    //处理进度百分比
}
-(void)onCompletion:(AliMediaDownloader *)downloader {
    //下载成功
}
```

> 说明 下载进度与处理进度的回调区别：下载进度完成前，是需要进行网络交互的；处理进度回调时，无需网络交互。

3. 准备下载源。

   通过preapre方法准备下载源。下载源支持AVPVidStsSource和AVPVidAuthSource两种。以AVPVidStsSource举例。示例如下：

```
//创建VidSts
AVPVidStsSource* stsSource = [[AVPVidStsSource alloc] init];
stsSource.vid = source.vid;//视频vid
stsSource.region = DEFAULT_SERVER.region;//接入区域
stsSource.securityToken = DEFAULT_SERVER.securityToken;//安全token
stsSource.accessKeySecret = DEFAULT_SERVER.accessKeySecret;//临时akSecret
stsSource.accessKeyId = DEFAULT_SERVER.accessKeyId;//临时akId
//准备下载源
[downloader prepareWithVid:stsSource];
```

4. 准备下载源。

   准备成功后，会回调onPrepared方法。选择某个Track进行下载，示例如下：

```
-(void)onPrepared:(AliMediaDownloader *)downloader mediaInfo:(AVPMediaInfo *)info {
    NSArray<AVPTrackInfo*>* tracks = info.tracks;
    //比如：下载第一个TrackInfo
    [downloader selectTrack:[tracks objectAtIndex:0].trackIndex];
}
```

5. 更新下载源并开始下载。

   完成以上操作步骤后，开始下载。示例如下：

```
//更新下载源
  [downloader updateWithVid:vidSource]
//开始下载
  [downloader start];
```

> 说明 为了防止VidSts和VidAuth过期，请更新下载源的信息。

6. 下载成功或失败后，释放下载器。

   下载成功后，调用release释放下载器。在onCompletion或者onError回调中。示例如下：

```
[self.downloader destroy];
self.downloader = nil;
```

> 说明 如果需要重新下载，请重新创建下载器。

9.3. 短视频列表播放器

您可以通过阿里云SDK使用短视频列表播放器，本文提供了搭建Android系统与iOS系统短视频列表播放器的操作步骤、示例及说明。

效果示例

> 注意 目前列表播放器只对MP4的视频做了优化，播放HLS视频不建议使用此列表播放器，而是建议使用多个播放器的实现方式。

Android端关键实现

1. 创建列表播放器。

   创建列表播放器，并设置相关参数。示例如下：

```
private AliListPlayer mVideoListPlayer;
……
//创建播放器（必须）
mVideoListPlayer = AliPlayerFactory.createAliListPlayer(mContext);
//开启硬解
mVideoListPlayer.enableHardwareDecoder(true);
//设置播放器参数
PlayerConfig config = mVideoListPlayer.getConfig();
//停止之后清空画面。防止画面残留（建议设置）
config.mClearFrameWhenStop = true;
mVideoListPlayer.setConfig(config);
//开启循环播放。短视频一般都是循环播放的
mVideoListPlayer.setLoop(true);
//开启自动播放。
mVideoListPlayer.setAutoPlay(true);
//设置预加载个数。此时会加载3个视频。（当前播放、已经播放、未播放）
mVideoListPlayer.setPreloadCount(1);
```

2. 添加多个播放数据源。

   列表播放器支持URL、点播Vid的数据源（STS方式）。示例如下：

```
//使用点播服务的视频源
 mVideoListPlayer.addVid("点播视频vid", "视频唯一标识");
//或者直接使用URL
mVideoListPlayer.addUrl("视频URL", "视频唯一标识");
```

> 注意 在列表播放器内部，视频是通过视频唯一标识区分的。如果添加的视频源的唯一标识一样，会导致播放错乱。请确保每次添加数据源的视频唯一标识都不一样。

3. 开始播放视频。

   播放器提供了以下接口，实现基本的定位播放功能。示例如下：

```
//直接跳转到某个视频播放
mVideoListPlayer.moveTo("视频唯一标识", STS信息) //点播Vid源（STS方式）
mVideoListPlayer.moveTo("视频唯一标识") //URL源
//播放上一个视频
mVideoListPlayer.moveToPrev("视频唯一标识", STS信息) //点播Vid源（STS方式）
mVideoListPlayer.moveToPrev("视频唯一标识") //URL源
//播放下一个视频
mVideoListPlayer.moveToNext("视频唯一标识", STS信息) //点播Vid源（STS方式）
mVideoListPlayer.moveToNext("视频唯一标识") //URL源
```

4. TextureView与播放器的配合使用。

   多数列表播放使用了TextureView。正确的使用，可以避免退后台返回黑屏的问题，优化播放体验。

   i. TextureView的基本设置。示例如下：

```
TextureView mTextureView = null;
SurfaceTexture mSurfaceTexture = null;
Surface mSurface = null;
……
mTextureView = findViewById(R.id.textureView);
mTextureView.setSurfaceTextureListener(new TextureView.SurfaceTextureListener()
{
            @Override
            public void onSurfaceTextureAvailable(SurfaceTexture surfaceTexture,
int width, int height) {
              //缓存SurfaceTexture，在退后台再进时使用
                mSurfaceTexture = surfaceTexture;
                mSurface = new Surface(surface);
                //设置播放的surface
                    mVideoListPlayer.setSurface(mSurface);
              //重绘界面，立即刷新界面
                    mVideoListPlayer.redraw();
            }
            @Override
            public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int
width, int height) {
              //画面大小变化的时候重绘界面，立即刷新界面
                    mVideoListPlayer.redraw();
            }
            @Override
            public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
              //这里一定要return false。这样SurfaceTexture不会被系统销毁
                return false;
            }
            @Override
            public void onSurfaceTextureUpdated(SurfaceTexture surface) {
            }
        });
```

ii. 退后台重进界面

在onResume中处理退后台再进的情况。示例如下：

```
public void onResume() {
      ……
      if (mVideoListPlayer != null && mSurface != null) {
            mVideoListPlayer.setSurface(mSurface);
            mVideoListPlayer.redraw();
      }
      ……
}
```

5. 封面的隐藏时机

播放视频时，隐藏列表播放封面隐藏。示例如下：

```
mVideoListPlayer.setOnRenderingStartListener(new IPlayer.OnRenderingStartListener()
{
            @Override
            public void onRenderingStart() {
                //视频第一帧开始渲染。此时可以隐藏封面
            }
        });
```

6. 销毁播放器

播放器使用完后，需要释放播放器，否则会有内存泄漏。示例如下：

```
//释放surface
mVideoListPlayer.setSurface(null);
//如果按照第4步使用了TextureView，那么还需要释放缓存的surface
if (mSurface != null) {
            mSurface.release();
}
if (mSurfaceTexture != null) {
            mSurfaceTexture.release();
}
//释放播放器
mVideoListPlayer.release();
```

iOS端关键实现

1. 创建列表播放器

创建列表播放器，并设置相关参数。示例如下：

```
//创建播放器（必须）
self.listPlayer = [[AliListPlayer alloc]init];
//开启硬解
self.listPlayer.enableHardwareDecoder = YES;
//循环播放
self.listPlayer.loop = YES;
//自动播放
self.listPlayer.autoPlay = YES;
//渲染模式
self.listPlayer.scalingMode = AVP_SCALINGMODE_SCALEASPECTFIT;
//代理设置
self.listPlayer.delegate = self;
//预加载的清晰度
self.listPlayer.stsPreloadDefinition = @"FD";
//显示view
self.listPlayer.playerView = self.simplePlayScrollView.playView;
//设置预加载个数。此时会加载3个视频。当前播放的，和上下1个。
mVideoListPlayer.setPreloadCount(1);
```

2. 添加多个播放数据源

   列表播放器支持URL、点播Vid的数据源（STS方式）。示例如下：

```
//使用点播服务的视频源
[self.listPlayer addVidSource:点播视频vid uid:"视频唯一标识"];
//或者直接使用URL
[self.listPlayer addUrlSource:"视频URL" uid: "视频唯一标识"];
```

> 注意 在列表播放器内部，视频是通过视频唯一标识区分的。如果添加的视频源的唯一标识一样的话，会导致播放错乱。请确保每次添加数据源的视频唯一标识都不一样。

3. 开始播放视频

   播放器提供了以下接口，实现基本的定位播放功能。示例如下：

```
//直接跳转到某个视频播放
[self.listPlayer moveTo:"视频唯一标识"]; //URL源
[self.listPlayer moveTo:"视频唯一标识" accId:accId accKey:accKey token:token
region:region]; //点播Vid源（STS方式）
//播放上一个视频
[self.listPlayer moveToPrev:"视频唯一标识"]; //URL源
[self.listPlayer moveToPrev:"视频唯一标识" accId:accId accKey:accKey token:token regi
on:region]; //点播Vid源（STS方式）
//播放下一个视频
[self.listPlayer moveToNext:"视频唯一标识"]; //URL源
[self.listPlayer moveToNext:"视频唯一标识" accId:accId accKey:accKey token:token regi
on:region]; //点播Vid源（STS方式）
```

4. 滑动控制操作

i. 点击暂停和播放。示例如下：

```
- (void)tap {
    if ([self.delegate
respondsToSelector:@selector(AVPSimplePlayScrollViewTapGestureAction:)]) {
        [self.delegate AVPSimplePlayScrollViewTapGestureAction:self];
    }
}
- (void)AVPSimplePlayScrollViewTapGestureAction:(AVPSimplePlayScrollView
*)simplePlayScrollView {
    if (self.playerStatus == AVPStatusStarted) {
        //如果是播放，则暂停
        [self.listPlayer pause];
    }else if (self.playerStatus == AVPStatusPaused) {
        //如果是暂停，则播放
        [self.listPlayer start];
    }
}
```

ii. 滑动控制

如果列表滑动速度较快，滑动的位置超过一个，则直接调用moveTo接口列表滑动后的回调中去判定。示例如下：

```
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    CGFloat indexFloat = scrollView.contentOffset.y/self.frame.size.height;
    NSInteger index = (NSInteger)indexFloat;
    .......
      //往下滑动一格
       if (index - self.currentIndex == 1) {
            if ([self.delegate
respondsToSelector:@selector(AVPSimplePlayScrollView:motoNextAtIndex:)]) {
                [self.delegate AVPSimplePlayScrollView:self
motoNextAtIndex:index];
            }
        }
        //往上滑动一格
        else if (self.currentIndex - index == 1){
            if ([self.delegate
respondsToSelector:@selector(AVPSimplePlayScrollView:motoPreAtIndex:)]) {
                [self.delegate AVPSimplePlayScrollView:self
motoPreAtIndex:index];
            }
        }
        //滑动多格
        else {
            if ([self.delegate
respondsToSelector:@selector(AVPSimplePlayScrollView:scrollViewDidEndDecelerating
Index:)]) {
                [self.delegate AVPSimplePlayScrollView:self
scrollViewDidEndDeceleratingAtIndex:index];
            }
        }
}
/**
 滚动事件,移动位置超过一个
 @param simplePlayScrollView simplePlayScrollView
 @param index 移动到第几个
 */
- (void)AVPSimplePlayScrollView:(AVPSimplePlayScrollView *)simplePlayScrollView
scrollViewDidEndDeceleratingAtIndex:(NSInteger)index {
    //找到第几个
    AVPDemoResponseVideoListModel *model = [self findModelFromIndex:index];
    //MoveTo到当前
    [self moveToCurrentModel];
}
```

5. **封面的隐藏时机**

   播放视频时，隐藏列表播放封面隐藏。示例如下：

```
-(void)onPlayerEvent:(AliPlayer*)player eventType:(AVPEventType)eventType {
    switch (eventType) {
        case AVPEventPrepareDone: {
        }
            break;
        case AVPEventFirstRenderedStart: {
            //隐藏封面图
            [self.simplePlayScrollView showPlayView];
        }
            break;
        default:
            break;
    }
}
```

6. 销毁播放器

   播放器使用完后，需要释放播放器，否则会有内存泄漏。示例如下：

```
[self.listPlayer stop];
[self.listPlayer destroy];
self.listPlayer = nil;
```

9.4. 使用诊断工具

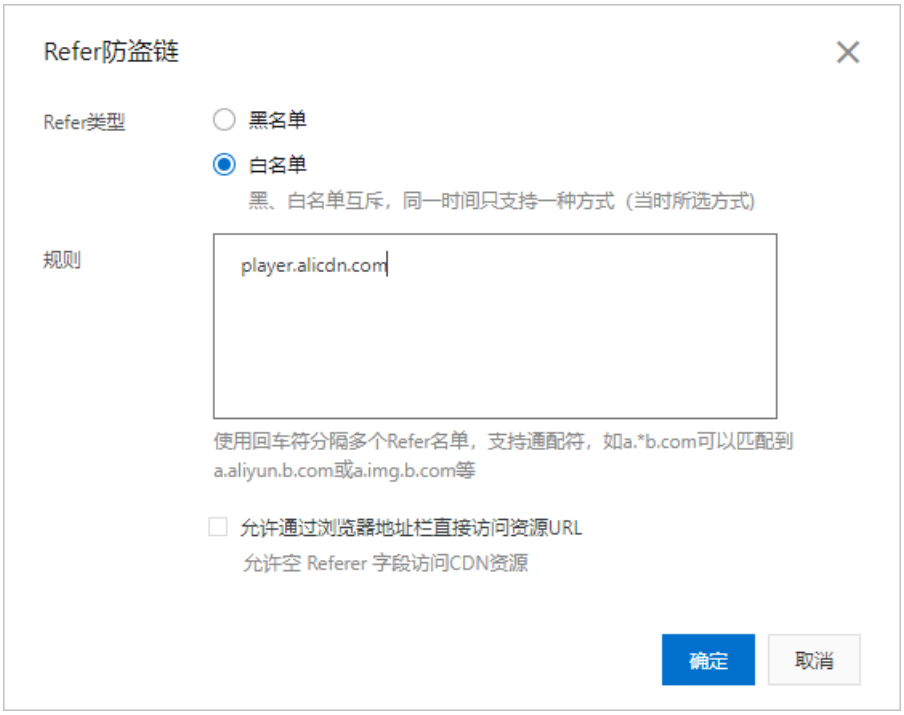
通过阅读本文，您可以了解如何打开Aliplayer播放器提供的诊断工具，并开始视频诊断。

前提条件

为了视频播放诊断更准确，需要进行以下设置：

- 开启视频播放域名的Refer防盗链，并将 `player.alicdn.com` 域名添加到白名单。详细操作，请参见 配置IP白名单。

> **说明**
>
> `player.alicdn.com` 为Aliplayer播放器的固定地址，请务必将此地址加入到白名单。



- 在视频播放域名的HTTP消息头，添加允许跨域访问的Access-Control-Allow-Origin。详细操作，请参见 关于跨域访问配置说明。

操作步骤

1. 打开诊断工具。
   您可以通过以下两种方式打开Aliplayer播放器的诊断工具：
   - 当播放视频出错时，单击 诊断，跳转到诊断工具页面 。

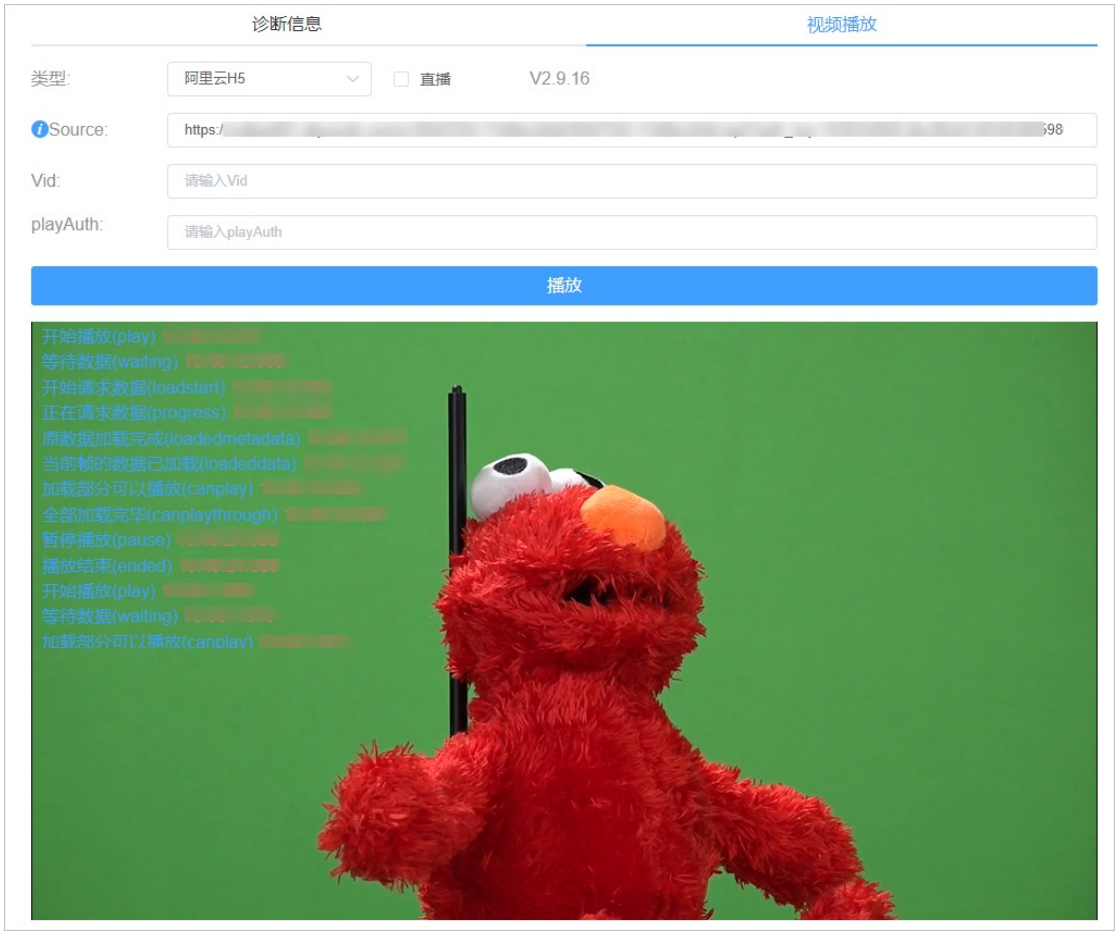
   - 直接访问诊断工具：诊断工具。
2. 基础信息诊断。

在诊断信息页，将显示用户环境的操作系统、浏览器、IP地址和运营商等。

3. 视频播放诊断。
   i. 在视频播放页，选择类型，即视频的播放方式，提供了原生H5、阿里云H5、阿里云Flash这三种播放方式。
   ii. 填写Source参数（视频播放地址），单击播放，即可开始诊断。如果没有Source参数，也可填写Vid和playAuth参数。

> **说明**
> Source参数的优先级最高。

iii. **开始诊断时，播放页面会显示播放日志。**



相关文档

错误码

10.错误码查询

本文提供了播放器SDK错误码类型的错误定义、错误值、错误含义并进行了说明，如果您在实际操作当中遇到返回错误码问题，可查询各类型错误码解决。

请求错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_NO_RESPONSE	20010001	536936449	服务器返回数据为空。
ERROR_SERVER_WRONG_JSON	20010002	536936450	服务器返回数据不为JSON格式。
ERROR_NO_MATCH_QUALITY	20010003	536936451	没有找到匹配的清晰度。
ERROR_PLAYAUTH_WRONG	20010004	536936452	PlayAuth解析错误。
ERROR_REQUEST_FAIL	20010005	536936453	请求失败。

POP错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_POP_UNKNOWN	20010100	536936704	POP未知错误。POP错误消息请参见错误码表。
ERROR_SERVER_POP_MISSING_PARAMETER	20010101	536936705	缺少参数。
ERROR_SERVER_POP_INVALID_PARAMETER	20010102	536936706	参数无效。
ERROR_SERVER_POP_OPERATION_DENIED	20010103	536936707	账号未开通视频点播服务。
ERROR_SERVER_POP_OPERATION_SUSPENED	20010104	536936708	账号已欠费，请充值。
ERROR_SERVER_POP_FORBIDDEN	20010105	536936709	无权限执行该操作。
ERROR_SERVER_POP_INTERNAL_ERROR	20010106	536936710	后台发生未知错误。
ERROR_SERVER_POP_SERVICE_UNAVALIABLE	20010107	536936711	服务不可用。
ERROR_SERVER_POP_SIGNATUREANONCE_USED	20010108	536936712	签名已经被使用。
ERROR_SERVER_POP_SECURITYTOKEN_MAILFORMED	20010109	536936713	安全Token不对。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_POP_SECURITYTOKEN_MISMATCH_ACCESSKEY	2001010A	536936714	安全Token与AccessKey不匹配。
ERROR_SERVER_POP_SIGNATURE_NOT_MATCH	2001010B	536936715	签名校验不对。
ERROR_SERVER_POP_ACCESSKEYID_NOT_FOUND	2001010C	536936716	没有找到AccessKeyId。
ERROR_SERVER_POP_TOKEN_EXPIRED	2001010D	536936717	Token过期。

点播错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_VOD_UNKNOWN	20010200	536936960	VOD未知错误。请参见 获取音视频播放地址。
ERROR_SERVER_VOD_FORBIDDEN_ILLEGALSTATUS	20010201	536936961	视频状态无效。
ERROR_SERVER_VOD_INVALIDVIDEO_NOTFOUND	20010202	536936962	视频不存在。
ERROR_SERVER_VOD_INVALIDVIDEO_NOSTREAM	20010203	536936963	根据您的筛选条件找不到可以播放的转码输出流。
ERROR_SERVER_VOD_FORBIDDEN_ALIYUNVODENCRYPTION	20010204	536936964	当前仅存在阿里云视频加密的转码输出流，必须使用阿里云播放器进行播放或者设置请求参数ResultType值为**Multiple**。
ERROR_SERVER_VOD_INVALIDAUTH_MEDIAID	20010205	536936965	AuthInfo与vid不一致。
ERROR_SERVER_VOD_INVALIDAUTHINFO_EXPIRETIME	20010206	536936966	AuthInfo过期。

MPS错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_MPS_UNKNOWN	20010300	536937216	MPS未知错误。
ERROR_SERVER_MPS_INVALID_MEDIAID	20010301	536937217	MediaId无效。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_MPS_INVALID_AUTHTIMEOUT	20010302	536937218	AuthTimeout无效。
ERROR_SERVER_MPS_INVALID_FORMATS	20010303	536937219	Formats无效。
ERROR_SERVER_MPS_INVALID_AUTHINFO	20010304	536937220	AuthInfo无效。
ERROR_SERVER_MPS_SIGNATURE_CHECK_FAILED	20010305	536937221	签名校验失败。
ERROR_SERVER_MPS_MEDIAID_NOT_EXIST	20010306	536937222	MediaId不存在。
ERROR_SERVER_MPS_MEDIA_RESOURCE_NOT_EXIST	20010307	536937223	媒体资源不存在。
ERROR_SERVER_MPS_MEDIA_NOT_PUBLISHED	20010308	536937224	媒体没有发布。
ERROR_SERVER_MPS_MEDIA_NOT_ENCRYPTED	20010309	536937225	媒体没有加密。
ERROR_SERVER_MPS_INVALID_CIPHERTEXTBLOB	2001030A	536937226	ciphertextblob无效。
ERROR_SERVER_MPS_CIPHERBLOB_NOT_EXIST	2001030B	536937227	CipherTextBlob不存在。
ERROR_SERVER_MPS_INTERNAL_ERROR	2001030C	536937228	服务器内部错误。
ERROR_SERVER_MPS_INVALID_IDENTITY_NOT_ORDER_VIDEO_SERVICE	2001030D	536937229	请求标识不允许操作。
ERROR_SERVER_MPS_UPDATE_CDN_DOMAIN_CONFIGS_FAIL	2001030E	536937230	更新主机配置失败。
ERROR_SERVER_MPS_AUTH_KEY_EXIST	2001030F	536937231	auth密钥已经存在。
ERROR_SERVER_MPS_AUTH_KEY_NOT_EXIST	20010310	536937232	auth密钥不存在。
ERROR_SERVER_MPS_INVALID_PARAMETER_OUT_OF_BOUND	20010311	536937233	参数超出范围。
ERROR_SERVER_MPS_INVALID_PARAMETER	20010312	536937234	参数无效。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_MPS_INVALID_PARAMETER_NULL_VALUE	20010313	536937235	参数不能为null。
ERROR_SERVER_MPS_INVALID_PARAMETER_EMPTY_VALUE	20010314	536937236	参数不能为空。
ERROR_SERVER_MPS_MEDIA_RESOURCE_NOT_MATCH	20010315	536937237	媒体资源不匹配。
ERROR_SERVER_MPS_MEDIA_NOT_FOUND_CIPHERTEXT	20010316	536937238	没有找到MediaId的密文资源。
ERROR_SERVER_MPS_INVALID_PARAMETER_RAND	20010317	536937239	指定的参数Rand无效。
ERROR_SERVER_MPS_REDIS_POOL_IS_EMPTY	20010318	536937240	缓存连接池为空。
ERROR_SERVER_MPS_SIGNATURE_CHECK_MEDIA_FAILED	20010319	536937241	签名和媒体ID不匹配。
ERROR_SERVER_MPS_SIGNATURE_CHECK_EXPIREDTIME_FAILED	2001031A	536937242	指定的到期时间值已过期。
ERROR_SERVER_MPS_INVALID_SESSION_TIME	2001031B	536937243	指定的参数SessionTime应该是>0。
ERROR_SERVER_MPS_INVALID_END_USER_ID	2001031C	536937244	EndUserId长度不对。
ERROR_SERVER_MPS_INVALID_URL	2001031D	536937245	指定的参数LicenseUrl格式不正确。
ERROR_SERVER_MPS_HTTP_REQUEST_FAILED	2001031E	536937246	请求失败。
ERROR_SERVER_MPS_XML_FORMAT_ERROR	2001031F	536937247	xml格式出错。
ERROR_SERVER_MPS_SESSION_NOT_EXIST	20010320	536937248	Session不存在。
ERROR_SERVER_MPS_REGION_NOT_SUPPORTED_API	20010321	536937249	API不支持。
ERROR_SERVER_MPS_DRM_NOT_ACTIVATED	20010322	536937250	此区域未激活DRM，请联系我们。
ERROR_SERVER_MPS_DRM_AUTH_ERROR	20010323	536937251	DRM验证错误，请为此媒体添加授权。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_MPS_CDN_CONFIG_NOT_EXIST	20010324	536937252	OSS域不存在CDN域配置。

直播时移错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_SERVER_LIVESHIFT_UNKNOWN	20010400	536937472	时移未知错误。
ERROR_SERVER_LIVESHIFT_REQUEST_ERROR	20010401	536937473	时移请求失败。
ERROR_SERVER_LIVESHIFT_DATA_PARSER_ERROR	20010402	536937474	时移数据解析失败。

私有加密错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_TBDRM_UNKNOWN	0x20012000	536944640	私有加密未知错误。
ERROR_TBDRM_DEMUXER_UNIMPLEMENTED	20012001	536944641	私有加密解封装未实现。

ARTP错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_ARTP_UNKNOWN	0x20013000	536948736	Artp未知错误。
ERROR_ARTP_DEMUXER_UNIMPLEMENTED	0x20013001	536948737	Artp模块加载失败，请检查动态库。
ERROR_ARTP_LOAD_FAILED	0x20013002	536948738	Artp视频播放加载失败。
ERROR_ARTP_STREAM_ILLEGAL	0x20013003	536948739	Artp流地址非法。
ERROR_ARTP_STREAM_NOT_FOUND	0x20013004	536948740	Artp流不存在。
ERROR_ARTP_STREAM_STOPPED	0x20013005	536948741	Artp流已经断了。
ERROR_ARTP_PLAY_TIMEOUT	0x20013006	536948742	Artp起播超时。
ERROR_ARTP_SPSPPS_AACCONF_TIMEOUT	0x20013007	536948743	Artp SPS/PPS或AAC Conf接收超时。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_ARTP_ARTP_MEDIA_INFO_TIMEOUT	0x20013007	536948743	Artp SPS/PPS或AAC Conf接收超时。
ERROR_ARTP_PACKET_RECV_TIMEOUT	0x20013008	536948744	Artp接收音视频数据包超时。
ERROR_ARTP_MEDIA_PROBE_FAILED	0x20013009	536948745	Artp数据包连同性探测失败。

播放错误码

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_UNKNOWN_ERROR	2001FFFF	537001983	未知错误。
ERROR_DEMUXER_OPENURL	20030001	537067521	打开URL失败。
ERROR_DEMUXER_NO_VALID_STREAM	20030002	537067522	无效的流。
ERROR_DEMUXER_OPENSTREAM	20030003	537067523	打开流失败。
ERROR_LOADING_TIMEOUT	20030004	537067524	加载超时。
ERROR_DATASOURCE_EMPTYURL	20030005	537067525	数据源URL为空。
ERROR_DECODE_VIDEO	20040001	537133057	视频解码失败。
ERROR_DECODE_AUDIO	20040002	537133058	音频解码失败。
ERROR_NETWORK_UNKNOWN	20050000	537198592	未知的网络错误。
ERROR_NETWORK_UNSUPPORTED	20050001	537198593	协议不支持。
ERROR_NETWORK_RESOLVE	20050002	537198594	不能解析域名。
ERROR_NETWORK_CONNECT_TIMEOUT	20050003	537198595	网络连接超时。
ERROR_NETWORK_COULD_NOT_CONNECT	20050004	537198596	无法连接到服务器。
ERROR_NETWORK_HTTP_403	20050005	537198597	403错误。
ERROR_NETWORK_HTTP_404	20050006	537198598	404错误。
ERROR_NETWORK_HTTP_4XX	20050007	537198599	其他的4XX错误。

错误定义	16进制错误值	10进制错误值	错误含义
ERROR_NETWORK_HTTP_5XX	20050008	537198600	5XX的服务器错误。
ERROR_NETWORK_HTTP_RANGE	20050009	537198601	不支持range请求。
ERROR_NETWORK_HTTP_400	2005000A	537198602	400错误。
ERROR_CODEC_UNKNOWN	20060000	537264128	未知的解码错误。
ERROR_CODEC_VIDEO_NOT_SUPPORT	20060001	537264129	视频编码格式不支持。
ERROR_CODEC_AUDIO_NOT_SUPPORT	20060002	537264130	音频编码格式不支持。
ERROR_GENERAL_UNKNOWN	20080000	537395200	标准错误。
ERROR_GENERAL_EPERM	20080001	537395201	标准错误-1，操作不允许等。
ERROR_GENERAL_ENOENT	20080002	537395202	标准错误-2，文件不存在等。
ERROR_GENERAL_EIO	20080005	537395205	标准错误-5，IO错误等。
ERROR_UNKNOWN	0x2FFFFFFF	805306367	未知错误。

下载错误码

错误定义	16进制错误值	10进制错误值	错误含义
DOWNLOAD_ERROR_NOT_SELECT_ITEM	30010000	805371904	没有选择下载项。
DOWNLOAD_ERROR_NO_DOWNLOAD_ITEM	30010001	805371905	没有可用下载项。
DOWNLOAD_ERROR_STS_SOURCE_NULL	30010002	805371906	没有设置Sts源。
DOWNLOAD_ERROR_AUTH_SOURCE_NULL	30010003	805371907	没有设置Auth源。
DOWNLOAD_ERROR_AUTH_SOURCE_WRONG	30010004	805371908	Auth格式不对。
DOWNLOAD_ERROR_INVALID_ITEM	30010005	805371909	选中的下载项不对。
DOWNLOAD_ERROR_URL_CANNOT_REACH	30010006	805371910	URL无法连接。

错误定义	16进制错误值	10进制错误值	错误含义
DOWNLOAD_ERROR_NOT_SUPPORT_FORMAT	30010007	805371911	下载的格式不支持。
DOWNLOAD_ERROR_ENCRYPT_FILE_NOT_MATCH	30010008	805371912	加密校验文件不匹配。
DOWNLOAD_ERROR_DOWNLOAD_SWITCH_OFF	30010009	805371913	下载功能被关闭。
DOWNLOAD_ERROR_NET_ERROR	3001000A	805371914	网络出错。
DOWNLOAD_ERROR_NOT_SET_SAVE_DIR	3001000B	805371915	没有设置下载路径。
DOWNLOAD_ERROR_CANNOT_CREATE_SAVE_DIR	3001000C	805371916	无法创建下载目录。
DOWNLOAD_ERROR_NO_SPACE	3001000D	805371917	没有空间。
DOWNLOAD_ERROR_WRITE_ERROR	3001000E	805371918	写入文件出错。
DOWNLOAD_ERROR_ENCRYPT_ERROR	3001000F	805371919	解密失败。
DOWNLOAD_ERROR_FILE_NOT_EXIST	30010010	805371920	文件不存在。
DOWNLOAD_ERROR_CLEAN_INVALID_PARAM	30010011	805371921	删除文件参数无效。
DOWNLOAD_ERROR_CLEAN_WRONG_STATUS	30010012	805371922	删除文件状态不对。
DOWNLOAD_ERROR_GET_AES_KEY_FAIL	30010013	805371923	获取AES密钥失败。
DOWNLOAD_ERROR_ENCRYPTION_NOT_SUPPORT	30010014	805371924	加密方式不支持。