ALIBABA CLOUD

阿里云

视频点播

短视频SDK

文档版本：20220601

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.短视频SDK简介

1.1. 产品介绍

阿里云视频点播提供集视频录制、裁剪、编辑、拼接、上传等功能于一体的短视频SDK。本文提供阿里云短视频SDK的核心优势、应用场景和功能特性（由License控制）等信息。

什么是短视频SDK

阿里云短视频SDK是集短视频录制、裁剪、编辑、上传等功能于一体的开发者工具。短视频SDK不仅提供易用、稳定、统一的接口，而且开源产品级的UI界面。开发者可以根据自己的业务实现短视频相关功能，还可以基于开源的UI界面完成界面定制，实现个性化的二次开发。

核心优势

- 快速接入，成本经济

  提供产品级SDK，最快2小时接入，节省自行开发耗费的人力物力，助你快速实现App短视频功能。

- 接口简单，开放性强

  接口简单易用，开放性强，专业版（UI开源）可以根据业务自由定制UI。

- 功能齐备，应用广泛

  录制功能自带断点录制、实时滤镜、高效美颜（支持阿里云美颜特效SDK和FaceUnity美颜贴纸SDK）、人脸贴图接口功能，支持本地视频导入压缩裁剪，对视频添加主题模板、动图、字幕、音乐等高级功能。

- 迭代打磨，稳定可靠

  视频技术经1000+应用商用验证，稳定可靠。

应用场景

适用于娱乐，社交，亲子，教育，新闻资讯等行业的短视频拍摄制作、多样互动玩法的场景。

功能特性

本节仅介绍短视频SDK所提供的主要功能总述，更多更详细的功能分类及介绍，请参见功能列表。

- 界面

  提供默认UI，并提供完整的UI交互源码，可自定义UI。

- 视频录制

  支持多种合拍模式、断点录制、回删、点击拍摄、长按拍摄、普通美颜、实时滤镜、闪光灯、实时水印、摄像头切换、分辨率设定、自带人脸库实现人脸贴图、多路混音和变速等功能。同时专业版支持对接第三方人脸AR能力实现高级美颜、美肌能力。

- 视频编辑

  支持视频裁剪、添加音乐、多路拼接等基础编辑功能，同时支持在编辑界面添加滤镜（包括静态滤镜和动效滤镜）、转场、字幕（含普通字幕和气泡字幕）、贴图（支持动态和静态）、音频变声和涂鸦（支持画笔粗细调整，颜色调整和撤销）等特效编辑功能。

- 动图

  支持在编辑界面添加动图，可在任意时间点添加并支持时间调整。

- 主题模板

支持生成主题模板，实现在视频编辑时的添加MV和剪同款功能。

- 字幕

  支持在编辑界面添加普通字幕、气泡字幕、花字及翻转字幕。

- 草稿箱

  支持编辑后生成草稿，或从草稿箱中编辑视频，并提供了草稿列表，草稿云同步等功能。

License版本

短视频SDK按提供功能不同可分为基础版、标准版和专业版，版本由License控制。使用短视频SDK的前提之一是获取License授权，更多信息，请参考获取短视频SDK License。

功能列表

下表介绍短视频SDK所提供的详细功能及基础版、标准版、专业版的支持情况。表格中的✓表示支持，×表示不支持。

功能点		功能说明	基础版	标准版	专业版
界面	默认UI	SDK包含一套默认的UI，布局、交互、界面可二次开发。	✓	✓	✓
	自定义UI	提供完整的UI交互源码，用户可自定义UI界面。支持替换图标和背景颜色，或完全自定义。	✓	✓	✓
	分辨率与屏比	支持高清、标清、超高清拍摄，支持16：9、4：3、1：1多种屏比拍摄。	✓	✓	✓
	清晰度	可设定帧率、质量等级，或自定义GOP、码率生成不同清晰度的视频。	✓	✓	✓
	多段录制	支持断点拍摄和连续拍摄。	✓	✓	✓
	自定义时长	自定义最长和最短拍摄时长。	✓	✓	✓
	变速	支持慢速和快速录制。	✓	✓	✓
	拍摄控制	拍摄可控制：切画幅、切摄像头、手电筒、曝光度、焦距、对焦。	✓	✓	✓
	拍照	拍摄支持抓取当前采集画面并保存为图片。	✓	✓	✓
	背景音乐	支持录制界面添加音乐，音乐资源由第三方提供，有额外费用。	✓	✓	✓
	实时水印	支持在录制时添加水印。	✓	✓	✓

功能点		功能说明	基础版	标准版	专业版
视频录制	实时滤镜	拍摄预览时可实时切换滤镜，支持自定义滤镜。	✓	✓	✓
	实时特效	拍摄预览时可设置抖动、分屏等常见特效，支持自定义特效。	✓	✓	✓
	人脸贴纸	拍摄内置人脸识别功能，在人脸上覆盖贴纸挂件等效果。	×	×	✓
	人脸识别接口	拍摄支持对接第三方人脸，进行人脸贴纸操作。	×	×	✓
	自定义渲染	采集数据可通过回调进行自定义渲染。	✓	✓	✓
	基础美颜	拍摄实时美颜，平滑无极调整强度。	✓	✓	✓
	高级美颜	录制支持引入第三方美颜（支持阿里云简介和FaceUnity美颜贴纸SDK，有额外费用），包含磨皮、红润、美白，大眼瘦脸等效果。	×	×	✓
	视频合拍	和已生成的视频进行合拍，实现双画面。	×	✓	✓
	多路混音	支持在录制过程中多路音频实时混音。	×	✓	✓
	回声消除	支持在录制过程中录制源回声消除。	×	✓	✓
	音频降噪	支持在录制过程中实时降噪。	×	✓	✓
	View录制	指定View进行录制，生成视频。	×	✓	✓
	多源录制	摄像头拍摄、View录制，本地视频进行实时合成录制，生成视频。	×	✓	✓
	视频导入	支持本地视频快速导入后编辑。	✓	✓	✓
	照片裁剪	支持照片画面大小的裁剪，同时支持画面填充和画面裁剪。	✓	✓	✓
	视频裁剪	支持视频画面大小和时长裁剪，同时支持画面填充和画面裁剪。	✓	✓	✓

功能点		功能说明	基础版	标准版	专业版
视频编辑（基础编辑）	原比例裁剪	支持保持原始视频比例裁剪视频时长，更加快速。	✓	✓	✓
	视频转码	可设置编码、码率、GOP、帧率生成不同压缩比的视频。	✓	✓	✓
	按帧预览	移动时间线时，在预览窗口显示基准游标停留的帧图像。	×	✓	✓
	视频截图	基于当前时间线，渲染当前画面并保存图片。	×	✓	✓
	多路拼接	支持导入多路视频或图片，进行前后拼接，生成1个视频文件。	×	✓	✓
	多路混排	支持导入多路视频，指定位置与大小，生成具有多个画面的视频文件。	×	✓	✓
	画中画	支持在主轨视频上叠加视频，生成带有画中画效果的视频文件。	×	✓	✓
	背景音乐	支持将音乐合成到视频中，不受时间特效影响。音乐资源由第三方提供，有额外费用。	×	✓	✓
	本地配音	支持本地音频合成到视频中，支持快速、慢速等时间特效调节。	×	✓	✓
	时间特效	支持对视频进行快速、慢速、倒放、反复操作。	×	✓	✓
	画面调节	在编辑界面，调整画面的亮度、白平衡、锐度、暗角、对比度等。	×	✓	✓
	静态滤镜	在编辑界面添加lut滤镜或着色器脚本滤镜，实时切换滤镜。	×	✓	✓
	特效滤镜	在编辑界面添加灵魂出窍，幻影等特效滤镜。	×	✓	✓
	转场	支持视频和照片间添加移动、淡入淡出、百叶窗等效果。	×	✓	✓
	帧动画	支持视频及画面上的挂件（贴纸、字幕等）进行位移、缩放等动画，支持自定义动画。	×	✓	✓

视频编辑（特效编辑）		功能说明	基础版	标准版	专业版
	贴纸	在编辑界面添加静态图片，可在任意时间点添加并支持时间调整。	×	✓	✓
	涂鸦	支持画笔样式、尺寸和颜色调整。	×	✓	✓
	片尾水印	支持在视频末尾添加片尾水印效果，可定义持续时间。	×	✓	✓
	自定义渲染	解码的帧数据可通过回调进行自定义渲染。	×	✓	✓
	音频降噪	支持对音频进行降噪处理。	×	✓	✓
	音频淡入淡出	支持对音频进行淡入淡出处理。	×	✓	✓
	音频静音	支持消除当前视频的原音和音乐声音。	×	✓	✓
	音频变声	在编辑界面将视频原音变成萝莉、大叔音等。	×	✓	✓
动图	动图	在编辑界面添加图片序列（动图），可在任意时间点添加并支持时间调整。	×	×	✓
主题模板	MV	在编辑界面添加MV效果，切换MV。	×	×	✓
	剪同款	引导客户选择相应的视频或图片，根据设计好的效果组合进行视频的合成。	×	×	✓
字幕	普通字幕	可添加多个字幕，可以设定字幕的位置、大小、角度、颜色、字体，以及每个字幕的开始和结束的时间。	×	×	✓
	气泡字幕	为文字增加背景图，支持动画背景，设定文字边框。	×	×	✓
	花字	带有艺术效果的多种色彩的文字样式。	×	×	✓
	翻转字幕	组合动画，可根据模板控制文字的出入时间点及动画效果。	×	×	✓
	草稿	录制或编辑后生成草稿，下次可直接进入编辑。	×	✓	✓
	草稿列表	可加入到草稿列表。	×	✓	✓

草稿箱功能点		功能说明	基础版	标准版	专业版
	草稿云同步	支持草稿上传与下载。	×	✓	✓
其他	相册选择	支持从相册过滤视频，也支持视频时长过滤。	✓	✓	✓
	缩略图	支持指定时间点进行抽帧，生成缩略图，支持快速模式。	✓	✓	✓
	视频信息	提取视频常用信息，包括分辨率、帧率、编码格式等。	✓	✓	✓
	上传到点播	云点播支持媒资管理、云端转码、内容审核等功能。	✓	✓	✓
	专家支持	短视频SDK技术支持。	工单	工单	钉钉群

常见咨询类问题

当您需要咨询关于短视频相关问题以及日常使用中遇到的问题，可以咨询阿里云视频云专门为短视频SDK打造的智能机器人智能客服。

相关文档

- 获取短视频SDK License
- SDK下载
- Android短视频SDK Demo
- iOS短视频SDK Demo
- Android短视频SDK
- iOS短视频SDK

1.2. 基础概念

短视频SDK是阿里云开源的一款播放器组件，拥有视频强大的播放功能，支持视频录制、视频合拍、视频拼接等能力。本文介绍短视频SDK中涉及的一些基本概念，以便于您更好地理解短视频SDK。

License

短视频SDK服务需要开通License，开通方式请参见获取短视频SDK License。

> **说明** 开通License后，请确保提交package/bundle id和Android Studio/XCode中对应配置保持一致。

视频分辨率

视频分辨率指的是视频横向和纵向上的有效像素，理论上视频分辨率越高，图像越清晰。但分辨率越高也意味着文件越大，处理越耗时。移动端考虑到不同设备性能差异，建议设置分辨率720P及以下。常见的视频分辨率如下：

清晰度	1:1	3:4	9:16
480P	480*480	640*480	853*480
540P	540*540	720*540	960*540
720P	720*720	960*720	1280*720
1080P	1080*1080	1440*1080	1920*1080

> **说明** 不要直接使用屏幕像素值作为视频分辨率。

码率

码率又叫比特率，指的是每秒传送的比特（bit）数。单位为bps（Bit Per Second）。压缩视频的时候给这个视频指定码率参数，用以告诉视频编码器期望的压缩后视频的大小。在一定范围内，码率越高，视频越清晰，文件也越大。建议码率：

清晰度	建议码率
480P	1000000~2000000
540P	2000000~3000000
720P	2000000~4000000
1080P	2000000~6000000

帧率

视频帧率指的是每秒钟显示的图像帧数，单位Frame per Second(fps)。帧率越高，图像越流畅，文件也越大。建议视频帧率：25~30。

关键帧

帧是组成视频图像的基本单位，视频文件是由多个连续的帧组成。关键帧也叫I帧，它是帧间压缩编码里的重要帧，解码时仅用I帧的数据就可重构完整图像，I帧不需要参考其他画面而生成。关键帧可以做为随机访问（seek）的参考点，可以当成图像。

GOP

Group of Picture（以下简称GOP）顾名思义就是有一组帧组成的一个序列。一个GOP由关键帧开始，后面跟随者一组B帧和P帧。GOP过小，会导致I帧的比例增高，压缩比降低。GOP过大，会导致随机访问（seek）更耗时，同时，会导致倒播卡顿（倒播需要解码一个GOP才能播放视频帧）。SDK中GOP默认值为5，建议GOP值为5~30。

> **说明** 编辑模块实现视频倒播功能，如果导入视频GOP过大，需要先转码处理。

填充模式

当素材图片或视频的分辨率长宽比与导出视频分辨率长宽比不一致时，会涉及填充模式的选择。

SDK支持两种填充模式：

填充模式	处理方法
裁剪模式	保持长宽比，裁剪图片，只显示中间区域。
缩放模式	保持长宽比，使图片能完整显示，上下或左右填充颜色。

编码方式

编码方式有以下两种：

编码方式	编码详情
软编	使用CPU进行编码。软编可以配置的参数更丰富，同等码率下生成的视频更清晰；但是编码速度比较慢，CPU负载高，手机更容易发热。
硬编	使用非CPU硬件进行编码。硬编编码速度更快，CPU负载低；但清晰度比软编略差，部分安卓设备上可能存在适配性问题。

资源说明

SDK资源主要包括人脸识别模型资源、滤镜资源和动效滤镜资源。SDK资源可以保存到网络端，也可以直接打包到apk中。考虑到SDK下载包的大小，建议您将SDK资源保存到网络端，在启动App时下载。

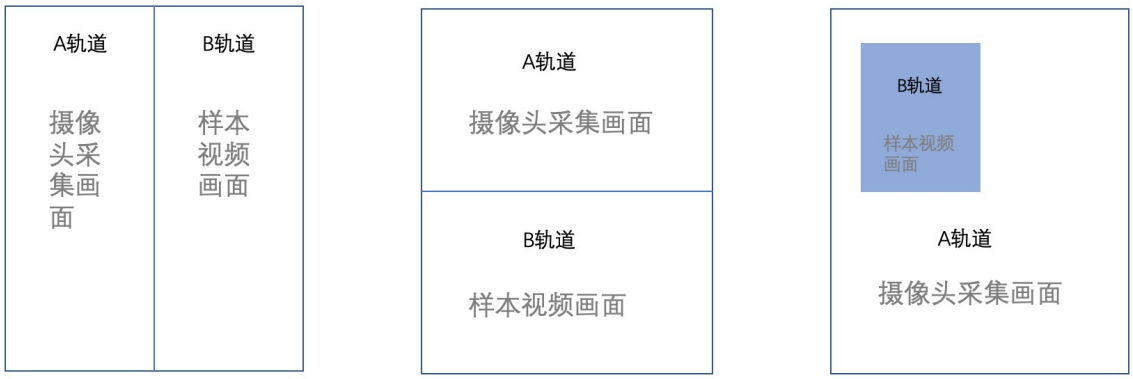
> **说明** SDK不支持assets流，如果是打包到apk中，启动后必须将资源拷贝到SD Card中。资源文件及使用说明可以在SDK下载包中获取。

支持格式

类型	格式
视频	MP4、MOV、FLV
音频	MP3、AAC、PCM
图片	JPG、PNG、GIF

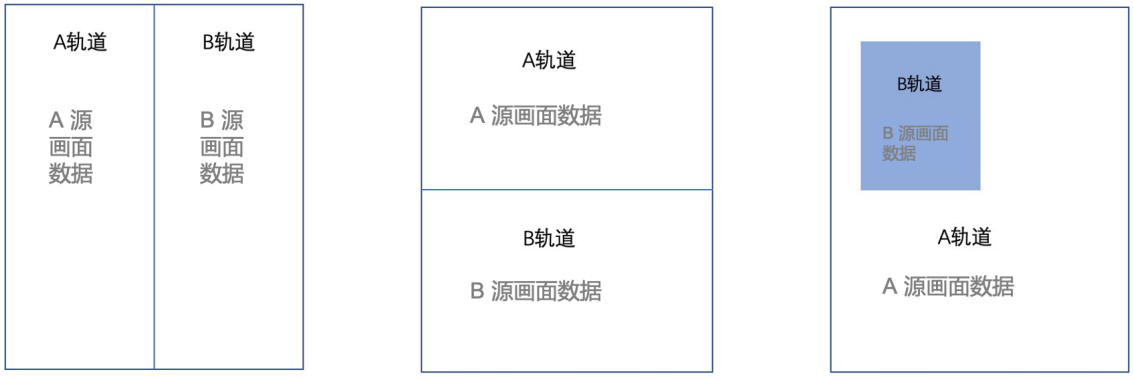
视频合拍

视频合拍从产品功能层面看，就是两路视频（一路来自样本视频，一路来自设备摄像头采集），按照指定的布局模式（左右分屏，上下分屏，画中画等）进行合成，合成出来的视频每一帧画面将会同时包含两路视频的画面，而合拍视频的音频部分则采用样本视频的音频。以下为范例视图，实际上SDK内部支持开发者自己组织布局，关于如何布局将在后面讲述。



多源录制

多源录制可以支持View录制、摄像头录制等多种视频采集源按需组合的合拍录制。从产品功能层面看，就是多个画面数据来源（例如View录制采集的画面数据、摄像头采集的画面数据），按照指定的布局模式（左右分屏、上下分屏、画中画等）进行合成，合成出来的视频每一帧画面将会同时包含上诉画面数据来源。以下为范例视图，实际上可支持开发者自己组织布局，关于如何布局将在后面讲述。



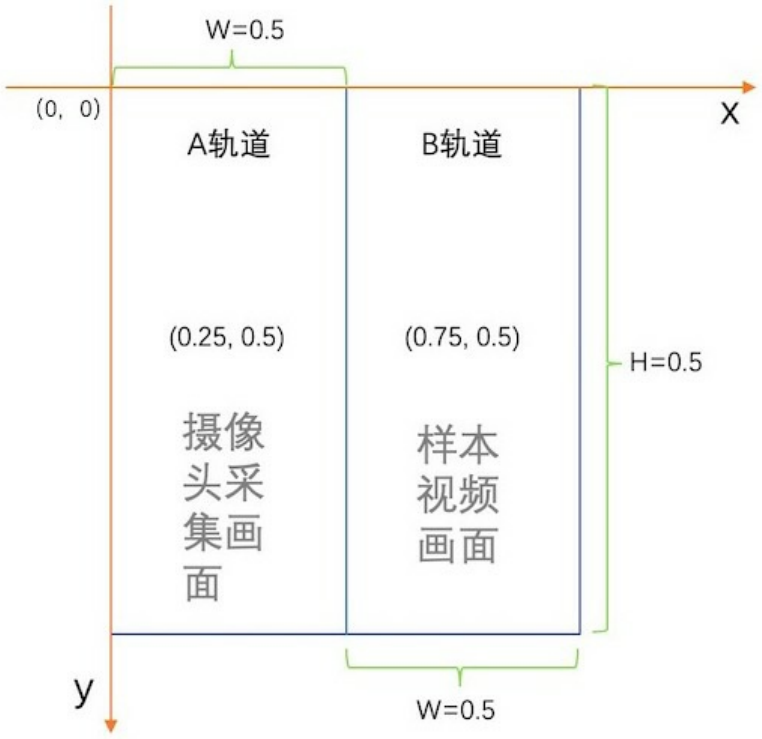
轨道

- 在上述视频合拍概念中提及的两路视频在SDK中被抽象为两个轨道：A轨道和B轨道，A轨道放设备采集的视频，B轨道放样本视频，用轨道抽象有利于开发者理解轨道布局的概念。
- 在上述多源录制中提及的多路画面数据来源在SDK中被抽象为多个轨道，如A轨道放摄像头采集画面，B轨道放View录制采集画面，用轨道抽象有利于开发者理解轨道布局的概念。

轨道布局

轨道布局是轨道的属性之一，用来描述该轨道的视频画面，在合拍生成的视频中如何“摆放”，轨道布局在一个归一化的坐标系中，从两个纬度来描述轨道布局信息，分别是中心点的坐标和轨道size（即宽高信息）。

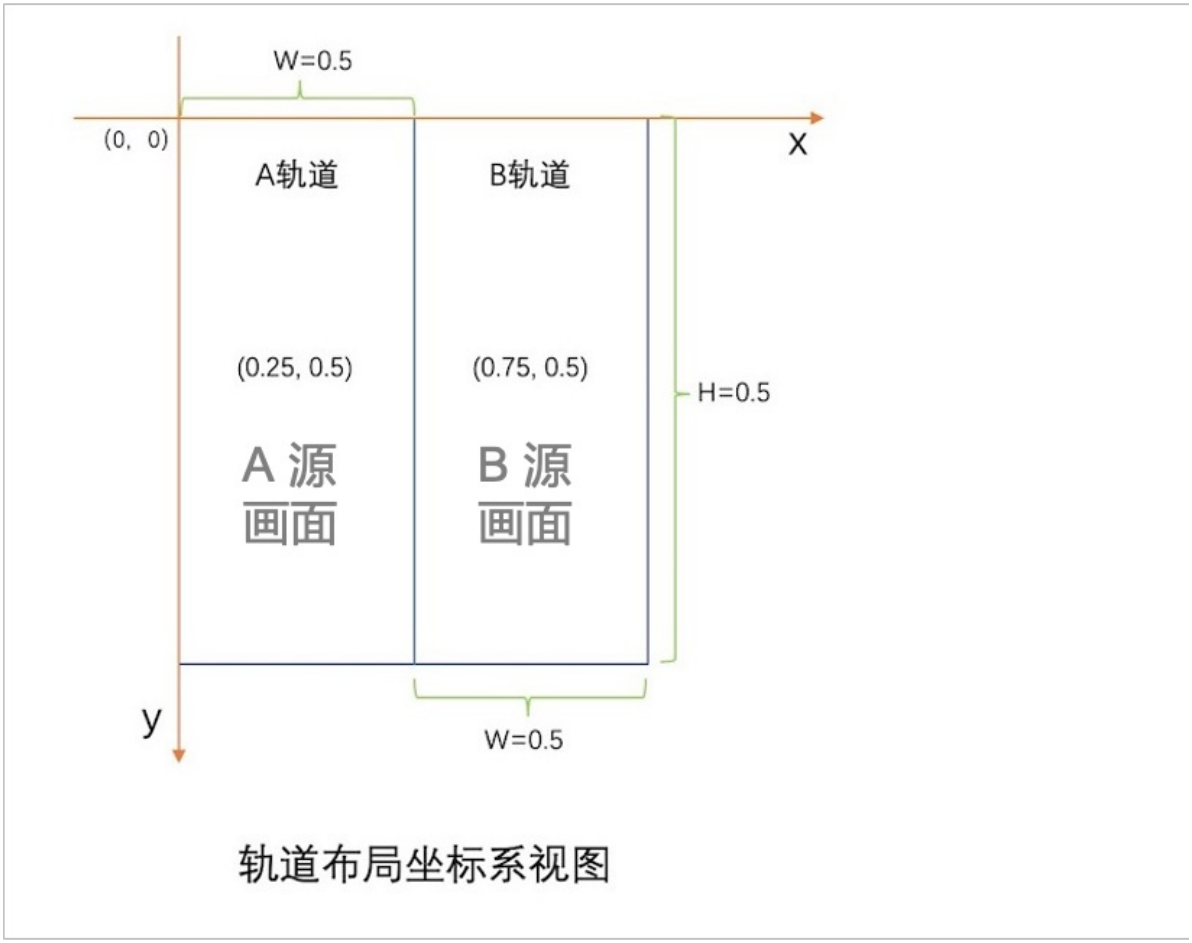
- 视频合拍的轨道布局如下图所示：



轨道布局坐标系视图

在该布局画面中，轨道A和轨道B的画面各占一半，因此，两个轨道的宽度均为0.5，而高度则都为1.0，而轨道A的中心点坐标：(0.25,0.5)，轨道B的中心点坐标：(0.75,0.5)。

- 多源录制的轨道布局如下图所示：



在该布局画面中，轨道A和轨道B的画面各占一半，因此，两个轨道的宽度均为0.5，而高度则都为1.0，而轨道A的中心点坐标：(0.25,0.5)，轨道B的中心点坐标：(0.75,0.5)。

2.获取短视频SDK License

短视频SDK使用License进行授权，本文为您介绍短视频SDK License的获取方式以及使用方式。

License升级说明

短视频SDK从3.29.0版本开始，接入一体化License服务。一体化License可为同一个阿里云账号下的所有APP提供视频云SDK的接入授权。服务升级的同时，短视频SDK的集成方式也有所变化。详情如下：

短视频SDK版本	集成方式
新接入	使用3.29.0及以上版本的新用户，按照官网帮助文档集成最新版本使用。 • Android端集成说明请参见：初始化SDK。 • iOS端集成说明请参见：初始化SDK。
SDK版本升级	如果您已接入低于3.29.0版本，在License有效期内，可以继续使用老版本SDK。如果您想升级到3.29.0及以后版本，集成时需要调用注册接口把LicenseKey和LicenseFile注入到SDK内。 **注意** LicenseKey和LicenseFile会通过授权邮件发送。如需补发，请通过开通License授权的步骤申请。 • Android端集成说明请参见：初始化SDK。 • iOS端集成说明请参见：初始化SDK。

申请试用License授权

请发送以下信息至指定邮箱videosdk@service.aliyun.com。收到申请邮件后，我们将于2个工作日之内处理完开通申请。

- 公司名称
- 应用名称
- 申请试用的SDK版本（基础、标准、专业）
- 联系人
- 联系电话
- 应用bundleID
- 包名和签名信息（MD5）
  - 要求：MD5格式、小写、无冒号。
  - 可使用签名获取工具来获取包名和签名信息，安装后按工具中的提示操作。
- 阿里云账号或UID

  如没有阿里云账号，请提前注册。注册步骤请参见注册阿里云账号。

> 注意
>
> - License免费试用期为一个月。
> - 请确保申请信息齐全、格式规范，后续若有需要更换包名、签名文件、BundleID任意一项，请重新申请。
> - 短视频专业版SDK的试用版支持高级美颜、智能抠图和手势识别等功能，这些功能由阿里云简介或第三方SDK提供。如需试用，需要在申请邮件中额外注明，美颜特效SDK或第三方SDK的授权证书会通过邮件的形式送达。

购买License授权

短视频SDK由License进行授权控制，不同的License版本价格不一样，支持的功能也不一样。

不同版本的License所支持的功能差异请参见功能列表，短视频SDK各版本的购买及说明请参见下表：

版本	购买地址	说明
基础版	流量包、存储包、转码包、智能审核包	• 购买点播流量包、存储包、转码包、智能审核包，**单日内订单总金额超过1900元时**，将赠送短视频SDK基础版1年License授权。 • 满足条件的客户，如需获取短视频SDK License，请参见开通License授权，通过邮件提供相应信息，以便为您开通短视频SDK License。 **说明** 每个订单可授权一个App（包括iOS和Android）。
标准版		• 购买点播流量包、存储包、转码包、智能审核包，**单日内订单总金额超过3万元时**，将赠送短视频SDK标准版1年License授权。 • 满足条件的客户，请参见开通License授权，通过邮件提供相应信息，以便为您开通短视频SDK License。 **说明** 每个订单可授权一个App（包括iOS和Android）。
专业版	短视频SDK专业版	• 计费方式为按功能模块组合计费。 • 购买后，请参见开通License授权，通过邮件提供相应信息，以便为您开通短视频SDK License。 • 短视频专业版SDK具备高级美颜、智能抠图和手势识别等功能，这些功能由阿里云美颜特效SDK或第三方SDK提供，需单独收费。美颜特效SDK的购买请参见简介；第三方SDK，可联系第三方商务人员获得折扣及其他优惠。 **说明** 1次购买最多支持3个App（最多支持10个马甲包），超出限制需额外购买。

开通License授权

购买或获赠短视频SDK后，请提供订单号、应用名称、应用BundleID、包名和签名信息（MD5格式、小写、无冒号）、阿里云账号或UID，并发送至邮箱videosdk@service.aliyun.com，以便为您开通短视频SDK License。收到邮件后，我们将于2个工作日之内为您开通License授权，如有紧急需求请联系商务人员。

> 注意
>
> - 从3.29.0版本开始会通过邮件发送LicenseKey和LicenseFile，您需要集成LicenseKey和LicenseFile到SDK内。
> - 确保提交申请的BundleID、包名、签名信息和自己工程中的完全一致。可使用签名获取工具来获取包名和签名信息，安装后按工具中的提示操作。
> - 测试时可以直接使用Demo提供的BundleID、包名、签名来体验。

续期License授权

短视频SDK License有使用期限，以3.29.0版本为界，License过期的判断标准不同。详情如下：

- 3.29.0以前版本：
  - 接口返回值为 `ALIVC_FRAMEWORK_LICENSE_FAILED(-10011001)` 。
  - 日志显示以下任一信息：
    - 30天无网络，License已禁用，请联系商务进行授权。
    - License已禁用，请联系商务进行授权。
    - License非法（包名和签名/BundleID没有在短视频SDK官网注册），超过7天试用，已经禁用。
- 3.29.0及以后版本：
  - 接口返回值为 `ALIVC_FRAMEWORK_LICENSE_FAILED(-10011001)` 。
  - 日志显示以下任一信息：
    - License已过期，请联系商务进行续费。
    - License未初始化，请参考文档进行接入。
    - License非法（包名和签名/BundleID没有在短视频SDK官网注册），请联系商务进行授权。
    - 使用增值服务：xxx已过期，请联系商务进行续费。（其中xxx指的是具体的增值服务。）
    - 使用增值服务：xxx非法，请联系商务进行授权。（其中xxx指的是具体的增值服务。）

> 说明 License失效后，请参见申请试用License授权、购买License授权或联系商务人员重新获得License授权。

3.短视频SDK Demo示例

3.1. Android短视频SDK Demo

本文为您提供Android端短视频SDK Demo的下载方式，并介绍Demo的目录说明及源码说明。

Demo下载

- 扫码下载Demo App

  您可以使用钉钉扫码下载短视频Demo进行体验。

- 链接下载Demo源码

  您可以在SDK下载页面的客户端SDK表格中，选择下载最新版短视频SDK及Demo源码。

Demo目录说明

以专业版DEMO为例，其目录结构如下：

```
├── JavaDoc.zip - API文档
├── ReleaseNote.md - 发布说明
├── ResourceProtocal - 资源协议
├── SDK - AAR包
└── demo - demo源码
```

Demo源码说明

```
├── AlivcMedia - 媒体选择
├── AliyunCrop - 视频裁剪
├── AliyunEditor - 视频编辑
├── AliyunFileDownLoader - 文件下载
├── AliyunRecorder - 视频录制（普通、合拍、多源录制）
├── FaceunityBeauty - Faceunity接入
├── QueenBeauty - Queen接入
├── app
```

3.2. iOS短视频SDK Demo

本文为您提供iOS短视频SDK Demo的下载方式。

Demo下载

- 扫码下载Demo App

  您可以使用钉钉扫码下载短视频Demo进行体验。

- 链接下载Demo源码

  您可以在SDK下载页面的客户端SDK表格中，选择下载最新版短视频SDK及Demo源码。

4.Android短视频SDK

4.1. 发布说明

V3.32.0

功能更新

- 视频合拍支持实时合成。
- 视频导出支持边合成边上传。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.32.0
标准版	com.aliyun.video.android:svideostand:3.32.0
基础版	com.aliyun.video.android:svideosnap:3.32.0

V3.31.0

功能更新

- 字幕背景新增圆角能力。
- 编辑模式新增镜像能力。
- 优化License校验逻辑。
- 优化开启日志落地能力，使接入时排查问题更精准。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.31.0
标准版	com.aliyun.video.android:svideostand:3.31.0
基础版	com.aliyun.video.android:svideosnap:3.31.0

V3.30.0

功能更新

- 优化License加载逻辑及权限管理。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.30.+
标准版	com.aliyun.video.android:svideostand:3.30.+

版本	Gradle依赖地址
基础版	com.aliyun.video.android:svideosnap:3.30.+

V3.29.0

功能更新

- 优化裁剪性能。
- License升级，初始化支持回调License授权信息。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖,且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.29.+
标准版	com.aliyun.video.android:svideostand:3.29.+
基础版	com.aliyun.video.android:svideosnap:3.29.+

V3.28.1

功能更新

修复多次切换转场效果加载耗时较长的问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.28.+
标准版	com.aliyun.video.android:svideostand:3.28.+
基础版	com.aliyun.video.android:svideosnap:3.28.+

V3.28.0

功能更新

- 优化功能，提高编辑导入效率。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.28.+
标准版	com.aliyun.video.android:svideostand:3.28.+
基础版	com.aliyun.video.android:svideosnap:3.28.+

V3.27.0

功能更新

- 新增剪同款功能。
- 多源录制支持回声消除、降噪、背景音乐与录音混音。
- 新增lut滤镜特效。
- 接口优化，统一SDK中接口的单位，时间：毫秒，角度：弧度。
- 修复HDR视频无法合成的问题。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.27.+
标准版	com.aliyun.video.android:svideostand:3.27.+
基础版	com.aliyun.video.android:svideosnap:3.27.+

V3.26.0

功能更新

- 优化SDK的稳定性问题。
- 修复部分音频格式不支持的问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.1 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.26.+
标准版	com.aliyun.video.android:svideostand:3.26.+
基础版	com.aliyun.video.android:svideosnap:3.26.+

V3.25.0

功能更新

- 新增画中画功能，支持在编辑界面添加画中画。
- 新增快速获取视频缩略图模式功能。
- 新增字幕动画功能，支持对字幕（花字）等做动画。
- 优化包的大小，集成后包体减少3M以上。
- 草稿箱新增自定义封面图。
- 修复部分已知问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
//AlivcFFmpeg必须依赖，且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
com.aliyun.video.android:AlivcFFmpeg:4.3.0-part // 短视频专用，包size较少。
com.aliyun.video.android:AlivcFFmpeg:4.3.0 //短视频与播放器共用。两个SDK同时接入，请用该版本。
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.25.+

版本	Gradle依赖地址
标准版	com.aliyun.video.android:svideostand:3.25.+
基础版	com.aliyun.video.android:svideosnap:3.25.+

V3.24.0

功能更新

- 优化SDK，删除FFmpeg软编码。
- 修复字幕在32位系统不生效的问题。
- 修复音频编码为HE-AACV2的视频，裁剪后无法在Chrome播放器播放的问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:4.3.0  //必须依赖4.3.0及以上版本
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.24.+
标准版	com.aliyun.video.android:svideostand:3.24.+
基础版	com.aliyun.video.android:svideosnap:3.24.+

V3.23.0

功能更新

- 新增草稿箱功能，支持导出草稿。
- 字幕新增背景色、对齐等功能。
- 合拍新增回声消除功能。
- Demo中美颜模块替换为Queen SDK。
- Demo中新增6个分屏滤镜特效。
- 多源录制支持SurfaceView录屏。
- 优化合拍性能，提升合成速度。

- 录制支持自动删除临时视频文件。
- SDK API增加注释，提高接入效率。
- 修复部分设备使用长视频合成至99%会失败的问题。
- 修复部分设备的拍摄黑屏等问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.1
```

版本	Gradle依赖地址
专业版	`com.aliyun.video.android:svideopro:3.23.+`
标准版	`com.aliyun.video.android:svideostand:3.23.+`
基础版	`com.aliyun.video.android:svideosnap:3.23.+`

V3.22.0

功能更新

- 编辑时新增花字功能。
- 新增局部屏幕采集功能。
- 新增边录屏边进行摄像头采集的功能。
- 在自定义特效Shader类中新增时间的内建变量（BUILTIN_PROGRESS）。
- 修复合拍时合拍视频高度可能会少两个像素的问题。
- 提升SDK稳定性，修复部分场景下不稳定的问题。

接口变动

- 新增接口：

  com.aliyun.svideosdk.editor.AliyunPasterManager#addCaptionWithStartTime

- 废弃接口：
  - com.aliyun.svideosdk.editor.AliyunPasterManager#addSubtitle
  - com.aliyun.svideosdk.editor.AliyunPasterManager#addSubtitleWithStartTime

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.1
```

版本	Gradle依赖地址
专业版	`com.aliyun.video.android:svideopro:3.22.0`
标准版	`com.aliyun.video.android:svideostand:3.22.0`
基础版	`com.aliyun.video.android:svideosnap:3.22.0`

V3.21.0

功能更新

- 新增合拍摄像头视频展示时，支持使用圆角边框。
- 支持使用HECI图片的导入工具。
- 修复部分机型进行软编过程中内存堆积并导致崩溃的问题。
- 修复自定义渲染回调相机矩阵没有及时更新的问题。
- 修复SDK稳定性问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	`com.aliyun.video.android:svideopro:3.21.0`
标准版	`com.aliyun.video.android:svideostand:3.21.0`

版本	Gradle依赖地址
基础版	com.aliyun.video.android:svideosnap:3.21.0

V3.20.0

功能更新

- 编辑模块音频增加淡入淡出效果。
- 编辑模块增加组合字幕功能。
- 编辑模块增加基础编辑能力。
- 修复部分机型多段视频素材编辑时，素材预览切换时出现花屏的情况。
- 修复编辑场景视频导出帧率设置不生效的问题。
- 修复Android平台自定义渲染时相机变化矩阵可能为空的问题。
- 修复SDK稳定性问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.20.0
标准版	com.aliyun.video.android:svideostand:3.20.0
基础版	com.aliyun.video.android:svideosnap:3.20.0

V3.19.0

功能更新

- 新增编辑模块音频降噪功能。
- 新增合拍和视频合并功能，支持设置背景图片和背景颜色。
- 新增合拍和视频合并功能，支持音轨合并。
- 新增支持录制预览阶段，回调音频数据。
- 修复编辑字幕功能，放大字体到某个字号，emoji图案不显示的问题。

- 修复设置水印、图片，添加某些透明光晕图片，光晕变色的问题。
- 修复添加静态图片旋转角度不对问题。

接口变动

确定不被引用的废弃接口，列表如下：

- com.aliyun.svideosdk.editor.AudioEffectType.EFFECT_TYPE_DENOISE
- com.aliyun.svideosdk.editor.AliyunIEditor.denoise(int, boolean)

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.19.0
标准版	com.aliyun.video.android:svideostand:3.19.0
基础版	com.aliyun.video.android:svideosnap:3.19.0

V3.18.1

功能更新

修复Android合拍非填充模式下花屏问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.18.1
标准版	com.aliyun.video.android:svideostand:3.18.1
基础版	com.aliyun.video.android:svideosnap:3.18.1

V3.18.0

功能更新

- 增加合拍视频指定使用的音轨功能（视频原音、录制声音、静音）。
- 修复android Q（10）切换画幅会闪烁黑边的问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.18.0
标准版	com.aliyun.video.android:svideostand:3.18.0
基础版	com.aliyun.video.android:svideosnap:3.18.0

V3.17.1

功能更新

- 修复某些机型合成后opengl导致的闪退问题。
- 修复自定义字体不生效问题。
- 修复AlivcSdkCore.setLogPath日志多线程问题。

其他

- **Maven集成方式仓库地址**

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

- **核心库**

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	`com.aliyun.video.android:svideopro:3.17.1`
标准版	`com.aliyun.video.android:svideostand:3.17.1`
基础版	`com.aliyun.video.android:svideosnap:3.17.1`

V3.17.0

功能更新

- 优化萝莉音效、新增方言音效。
- 修复拍照在极端场景下的闪退问题。

接口变动

- Android SDK包名重构优化，新包名统一以com.aliyun.svideosdk.*命名。

  详情请参考：API参考与辅助转换工具。
- 删除确定不被引用的废弃接口，列表如下：
  - com.error.NativeErrorCode。
  - com.qu.preview.callback.OnNativeReady。
  - com.aliyun.qupai.editor.AliyunIExporter。
  - com.aliyun.qupai.editor.AliyunIPlayer。
  - com.aliyun.qupai.editor.OnPlayCallback。
  - com.aliyun.qupai.editor.OnPreparedListener。
  - com.aliyun.querrorcode.AliyunVideoCoreError。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.17.0
标准版	com.aliyun.video.android:svideostand:3.17.0
基础版	com.aliyun.video.android:svideosnap:3.17.0

V3.16.2

功能更新

修复高斯模糊背景问题。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.16.2
标准版	com.aliyun.video.android:svideostand:3.16.2
基础版	com.aliyun.video.android:svideosnap:3.16.2

V3.16.1

功能更新

- 修复添加字幕后每行字数不统一问题。
- 修复字幕和动图动画，二次编辑时效果移动不正确问题。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版式	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.16.1
标准版	com.aliyun.video.android:svideostand:3.16.1
基础版	com.aliyun.video.android:svideosnap:3.16.1

V3.16.0

功能更新

- 恢复主流动画功能。
- 修复线上反馈偶现崩溃问题。
- 修复长视频可能出现的播放卡顿问题。
- 修复部分机型兼容性导致的录制崩溃问题。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.16.0
标准版	com.aliyun.video.android:svideostand:3.16.0

版本	Gradle依赖地址
基础版	com.aliyun.video.android:svideosnap:3.16.0

V3.15.0

功能更新

- 修复合成视频播放卡顿问题。
- 修复视频多段变速失效问题。
- 修复一些机型前置摄像头曝光区域无效问题。
- 新增基于自定义特效制作规范的两组转场、滤镜效果转场与滤镜效果。

接口变动

- 新增自定义特效参数调节接口，支持实时调节特效参数。
- 支持自定义滤镜、转场特效，自定义特效制作规范请参考官方文档。

其他

- Maven集成方式仓库地址
- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.3
com.aliyun.video.android:AlivcFFmpeg:2.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.15.0
标准版	com.aliyun.video.android:svideostand:3.15.0
基础版	com.aliyun.video.android:svideosnap:3.15.0

V3.14.0

功能更新

- 适配Android Q系统，提升Android Q系统录制编辑输出视频性能。
- 优化录制时实现，解决偶现的卡死问题。
- 修复已知几处内存泄漏并优化部分性能。

问题修复

- 修复几处接口偶现返回-10000004的问题。

- 修复部分异常视频偶现裁剪卡死问题。
- 修复录制同时拍照和对焦可能触发的死锁问题。
- 修复编辑设置背景颜色不生效问题。
- 修复几处内存泄漏和其他已知问题。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2
com.alivc.conan:AlivcConan:1.0.2
com.aliyun.video.android:AlivcSvideoFFmpeg:1.1.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.14.0
标准版	com.aliyun.video.android:svideostand:3.14.0
基础版	com.aliyun.video.android:svideosnap:3.14.0

V3.13.0

功能更新

- 录制模块稳定性，性能全面优化。
- 录制模块支持基于RACE的美颜美型功能。

接口变动

录制模块废弃mv接口，去除添加MV功能。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2 （对应AlivcCore.jar）
com.alivc.conan:AlivcConan:1.0.1
com.aliyun.video.android:AlivcSvideoFFmpeg:1.0.2
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.13.0
标准版	com.aliyun.video.android:svideostand:3.13.0
基础版	com.aliyun.video.android:svideosnap:3.13.0

V3.12.0

功能更新

- 增加日志分析功能

```
AlivcSdkCore#setDebugLoggerLevel(AlivcDebugLoggerLevel level)
```

提供三个等级供用户设置：

- AlivcDLAll全量log分析，只建议在定位疑难问题时开启，不建议在正式发版中使用。
- AlivcDLNormal能分析warning，error级别的日志，建议使用这个等级来做日志分析。
- AlivcDLClose关闭日志分析功能。

以上功能只会做SDK的日志分析。

- 编辑模块性能提升。

接口变动

- 编辑模块废除addRunningDisplayMode接口，去除动态切换内容模式功能。
- 编辑模块废除removeRunningDisplayMode接口，去除删除动态切换内容模式功能。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.2.2 （对应AlivcCore.jar）
com.alivc.conan:AlivcConan:1.0.1
com.aliyun.video.android:AlivcSvideoFFmpeg:1.0.2
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.12.0

版本	Gradle依赖地址
标准版	com.aliyun.video.android:svideostand:3.12.0
基础版	com.aliyun.video.android:svideosnap:3.12.0

V3.11.0

功能更新

- 提升片段录制起停的速度和录制合成的速度，分段录制更加流畅。
- 优化录制进度回调粒度和精准度。
- 精准控制gop，提升部分场景下的转码速度。

接口变动

- 对外错误码统一，所有错误码统一到AliyunErrorCode。
- 新增方法：String getErrorCodeMessage(int errorCode)，获取错误描述。

问题修复

- 修复缩略图取帧FILL模式不生效问题、抖音视频无法取帧显示问题。
- 修复编辑倒放后合成的视频首帧灰帧问题。
- 修复编辑暂时情况下添加涂鸦后，撤销不生效问题。
- 修复OpenH264、ffmpeg编码下，录制删除片段后切画幅后再录制花屏问题。
- 修复gif帧数解析不准确的问题。
- 修复特定视频倒播开始播放时卡顿问题。
- 修复多段录制输出视频音视频不同步。
- 修复录制生成视频时长不精准的问题。

其他

- Maven集成方式仓库地址

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

- 核心库

```
om.aliyun.video.android:core:1.2.1 （对应AlivcCore.jar）
com.alivc.conan:AlivcConan:0.9.5.1
com.aliyun.video.android:AlivcSvideoFFmpeg:1.0.1
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.11.0 -对应专业版的AliyunSdk-RCE.aar和armeabi-v7a&arm64-v8a的so库

版本	Gradle依赖地址
标准版	com.aliyun.video.android:svideostand:3.11.0 -对应标准版的AliyunSdk-RCE.aar和armeabi-v7a&arm64-v8a的so库
基础版	com.aliyun.video.android:svideosnap:3.11.0 -对应基础版的AliyunSdk-RC.aar和armeabi-v7a&arm64-v8a的so库

V3.10.5

功能更新

- 新增合拍功能接口AliyunIMixRecorder。
- 新增多轨道视频拼接（可以实现画中画，左右分屏等效果）AliyunIMixComposer。

V3.10.0

功能更新

- 编辑新增大魔王，小黄人音效。
- 编辑新增mjpeg视频格式支持。
- 编辑播放提升对部分损坏视频文件的兼容性。
- 编辑/转码新增对hevc视频硬解支持。
- 转码速度提升。
- 录制新增重新设置预览窗口大小接口AliyunIRecorder.resizePreviewSize。
- 新增合成及上传单独接口，可支持单独合成及单独上传。

问题修复

- 录制修复小段录制视频时长不准确问题。
- 优化一些句柄未释放导致的泄漏隐患问题。

接口变动

- 对外错误码统一，所有错误码统一到AliyunErrorCode。
- 新增方法：String getErrorCodeMessage(int errorCode)，获取错误描述。

其他

- Maven集成方式仓库地址

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

- 核心库

```
com.aliyun.video.android:core:1.1.2 （对应AlivcCore.jar）
com.alivc.conan:AlivcConan:0.9.4
com.aliyun.video.android:AlivcSvideoFFmpeg:1.0.0
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.10.0 -对应专业版的AliyunSdk-RCE.aar和armeabi-v7a&arm64-v8a的so库
标准版	com.aliyun.video.android:svideostand:3.10.0 -对应标准版的AliyunSdk-RCE.aar和armeabi-v7a&arm64-v8a的so库
基础版	com.aliyun.video.android:svideosnap:3.10.0 -对应基础版的AliyunSdk-RC.aar和armeabi-v7a&arm64-v8a的so库

V3.9.0

功能更新

- 提升编辑模块seek性能。
- 新增音效接口，提供萝莉，大叔，混响，回声四种音效。
- libAliFaceAREngine.so与libFaceAREngine.so合并为一个.so，只保留了libAliFaceAREngine.so。

接口变动

OnFrameCallBack接口回调触发线程变更为非主线程。

V3.8.0

功能更新

- 优化了编辑播放能力，流畅播放不卡顿。
- 优化了编辑合成的速度。
- 优化了视频录制预览清晰度。
- 提升低端机器上的录制帧率。
- 短视频SDK全面支持Maven依赖。

接口变动

- RecordCallback部分回调所在的线程有变动：
  - RecordCallback#onComplete：由主线程回调变为子线程回调，如果有UI操作，需要开发者将相关操作post主线程。
  - RecordCallback#onProgress：由主线程回调变为子线程回调，如果有UI操作，需要开发者将相关操作post主线程。
  - RecordCallback#onMaxDuration：由主线程回调变为子线程回调，如果有UI操作，需要开发者将相关操作post主线程。
  - RecordCallback#onError：由主线程回调变为子线程回调，如果有UI操作，需要开发者将相关操作post主线程。

  该改动主要是为了保证回调数据与SDK内部状态的一致性，减少异常问题。
- EditorCallback回调变动：

- EditorCallback由原先的Interface改为abstract class。
- 添加mNeedRenderCallback属性，该属性可以控制是否需要onCustomRender和onTextureRender两个回调，在关掉这两个回调的情况下，编辑模块的性能会有一定提升。目前默认为关闭状态。如果需要开启，请设置这个参数为：

```
mNeedRenderCallback = EditorCallBack.RENDER_CALLBACK_CUSTOM(开启onCustomRender);
mNeedRenderCallback = EditorCallBack.RENDER_CALLBACK_TEXTURE（开启onTextureRender);
mNeedRenderCallback = EditorCallBack.RENDER_CALLBACK_TEXTURE|EditorCallBack.RENDER_CALLBACK_CUSTOM（同时开启两个回调）；
```

其他

- Maven集成方式仓库地址

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

- 核心库

```
compile 'com.aliyun.video.android:core:1.1.0' -对应AlivcCore.jar
```

版本	Gradle依赖地址
专业版	com.aliyun.video.android:svideopro:3.8.0 -对应专业版的AliyunSdk-RCE.aar com.aliyun.video.android:svideopro-armv7a:3.8.0 -对应armeabi-v7a架构的短视频专业版所有so库 com.aliyun.video.android:svideopro-arm64:3.8.0 -对应arm64-v8a架构的短视频专业版所有so库
标准版	com.aliyun.video.android:svideostand:3.8.0 -对应标准版的AliyunSdk-RCE.aar com.aliyun.video.android:svideostand-armv7a:3.8.0 -对应armeabi-v7a架构的短视频标准版所有so库 com.aliyun.video.android:svideostand-arm64:3.8.0 -对应arm64-v8a架构的短视频标准版所有so库
基础版	com.aliyun.video.android:svideosnap:3.8.0 -对应基础版的AliyunSdk-RC.aar com.aliyun.video.android:svideosnap-armv7a:3.8.0 -对应armeabi-v7a架构的短视频基础版所有so库 com.aliyun.video.android:svideosnap-arm64:3.8.0 -对应arm64-v8a架构的短视频基础版所有so库

说明 短视频SDK内部不再包含上传SDK，开发者需要另外通过gradle添加外部依赖：compile 'com.aliyun.video.android:upload:1.5.2'。考虑到SDK稳定性监控和未来数据相关需求，短视频目前必须要依赖库：compile 'com.alivc.conan:AlivcConan:0.9.0'以及添加混淆，参考demo。

V3.7.8.1

接口变动

AliyunIRecorder新增postToGl和removeFromGl两个接口，用于向gl线程post和remove操作，一些需要依赖gl资源或者释放gl资源的操作可以通过这两个接口进行。

V3.7.8

功能更新

优化预览和录制的帧率，帧率有大幅提升。

接口变动

- `AliyunIRecorder.setDisplayView(GLSurfaceView surfaceView)` 变更为 `AliyunIRecorder.setDisplayView(SurfaceView surfaceView)` ，参数GLSurfaceView变更为SurfaceView。
- 自定义渲染（第三方渲染）销毁gl资源，以前GLSurfaceView时可以通过GLSurfaceView.queueEvent来做，现在增加了一个gl资源销毁的回调OnTextureIdCallBack.onTextureDestroyed()，需要统一在这里面做。
- 支持随意切换surface窗口大小，无需重启preview（如果要考虑重新选择采集分辨率则依然需要重启）。
- RecordCallback.onInitReady只会在AliyunIRecorder创建时回调一次（setRecordCallback），实际上只是为了保留老版本的兼容性，现在的版本AliyunIRecorder创建完即可进行相关操作，不需要等待onInitReady也可以。

V3.7.7

功能更新

增加AlivcSdkCore类，主要用于debug调试，AlivcSdkCore#register函数用于在debug模式下替换动态库，AlivcSdkCore#setLogLevel用于定制log等级。

其他

- 可以进入机器人答疑，可以输入关键字信息获取答案，请尽量输入准确的信息，如：接口文档，如何添加普通动图等。
- 提高合成，裁剪的清晰度。
- 整体稳定性提升。

V3.7.5

功能更新

- 修复编辑使用第三方渲染接口可能导致crash的bug。
- 时间特效播放流畅度提升。
- gif适配性扩展。
- 奇数分辨率导入视频支持。
- 优化多段录制音视频同步问题。
- 提升稳定性。

V3.7.0

功能更新

- 编辑预览播放增加replay接口，如果要重播，则需要在收到onEnd回调后调用replay，具体参考Demo代码。

- 修改静音接口AliyunIEditor#setAudioSilence的实现，现在静音接口只能够在预览播放时静音，如果要实现合成的视频静音，则需要使用AliyunIEditor#setVolume(0)，将输出音量设置为0。
- 编辑的AliyunPasterBaseView接口新增部分属性接口，主要为以下属性：

```
getTextMaxLines—获取最大行数
getTextAlign()—获取文字对齐方式
getTextPaddingX()—获取文字X轴距离左边的边距，以左上角为原点
getTextPaddingY()—获取文字Y轴距离上边的边距，以左上角为原点
getTextFixSize()—获取文字字号
getBackgroundBitmap()—获取文字背景图
isTextHasLabel()—是否有背景色
getTextBgLabelColor()—获取文字背景色以上接口需要开发者实现。
```

- AliyunIEditor#applySourceChange更新视频源后，不会自动播放，需要开发者控制播放，也就是说如果要继续播放则需要调用AliyunIEditor#play接口。
- 缩略图/取帧AliyunIThumbnailFetcher相关接口包名更换，可先预编译一遍，对于编译报错的，删除原有的import，然后重新import。
- 缩略图/取帧的回调，AliyunIThumbnailFetcher$OnThumbnailCompletion.onThumbnailReady()参数有所变动，原先的SharableBitmap改为Bitmap，可以直接使用，无需做回收。
- 缩略图/取帧的接口addVideoSource,addImageSource增加转场时间的参数，如果导入的视频需要考虑转场效果的时间，则需要设置转场时间，如果不需要则填0就可以了。
- 移除ScaleMode类，由VideoDisplayMode类代替。
- AliyunIReocder、AliyunICrop现在支持多实例，原有生成类AliyunRecorderCreator、AliyunCropCreator中的destroy方法被移除。
- libQuCore-ThirdParty.so由libsvideo_alivcffmpeg.so替代。
- 部分结构类包置为发生改变，如在原包名下找不到该类，请删除该类的import地址，重新import。
- 修复了部分Crash Bug。
- 修复了倒播卡顿的bug。
- 修复了部分机型动效滤镜效果不对的问题。
- 添加转场效果（TransitionBase），具体查看接口文档，同时AliyunIimport接口addVideo，addImage函数优化，去掉了原来的关于转场的inDuration，outDuration，overlapDuration三个参数，统一由TransitionBase的子类来提供更为丰富的转场效果。
- 添加特技效果接口AliyunIEditor#addFrameAnimation，支持自定义动画，具体查看接口文档。
- 导入多段视频支持添加多个变速时间特效（反复和倒放还是只支持单段视频的）。
- 新增删除变速效果的接口AliyunIEditor#deleteTimeEffect。
- 指定流、指定时间添加高斯模糊效果AliyunIEditor#applyBlurBackground。
- 指定流、指定时间设置显示模式填充/裁剪AliyunIEditor#addRunningDisplayMode。
- 增加配音接口，配音接口的音效跟随时间特效变动AliyunIEditor#applyDub。

其他

废弃录制添加mv的相关接口，包括int applyMv(EffectBean effectMv)void pauseMv()void resumeMv()void restartMv()废弃之后，相关接口可以继续使用，我们会在未来的某个版本彻底移除这些接口。

V3.6.5

功能更新

- 合成不支持ffmpeg软编。

- 添加时间特效会先走onEnd回调问题。
- 编辑设置音量，合成时设置的值无效，并且会放大音量，更改SDK默认音量值。
- 部分视频裁剪在99%卡住。
- 部分手机裁剪后的视频，编辑预览播放卡顿。
- 部分手机上特效滤镜有虚线。
- 部分手机上remove music Crash问题。
- 修复倒播卡顿的问题。
- 解决了yuv转rgb使用bt709公式造成的色域问题。
- 支持aac sbr格式音频。
- 音频采样率不对的问题。
- 修复了一些特效滤镜的适配性问题。
- 更新上传库，新增的字段需要短视频那边也要重新集成新增接口。

接口变动

增加了Alivc.jar，开发者的工程中需要加入对这个jar包的依赖。

V3.6.0

接口变动

- 多视频导入（AliyunIImport），添加视频、图片（addVideo、addImage）参数发生变化，原先的fadeDuration现在拆分了，拆分成上一个视频的出场时间（outDuration），下一个视频的入场时间（inDuration），以及两段视频出场入场的重合时间（overlapDuration）。
- 创建AliyunIEditor时参数变化AliyunEditorFactory.creatAliyunEditor(Uri uri, EditorCallBack callback)，原参数只有uri现在增加了EditorCallback，替代了之前的OnPlayCallback其中：

旧接口	对应新接口
OnPlayCallback.onPlayCompleted	EditorCallback.onEnd
OnPlayCallback.onError	EditorCallback.onError
OnPlayCallback.onTextureIDCallback	EditorCallback.onCustomRender
OnPlayCallback.onPlayStarted（去掉了）	OnPlayCallback.onSeekDone（去掉了）

- 创建播放器实例接口createAliyunPlayer()已不存在，播放器接口AliyunIPlayer也去掉了，其中播放控制对应的方法直接使用AliyunIEditor中的方法：

旧接口	对应新接口
AliyunIPlayer.getCurrentPosition	AliyunIEditor.getCurrentPlayPosition
AliyunIPlayer.getDuration	AliyunIEditor.getDuration
AliyunIPlayer.getRotation	AliyunIEditor.getRotation
AliyunIPlayer.getVideoHeight	AliyunIEditor.getVideoHeight
AliyunIPlayer.getVideoWidth	AliyunIEditor.getVideoWidth

旧接口	对应新接口
AliyunIPlayer.isAudioSilent	AliyunIEditor.isAudioSilense
AliyunIPlayer.isPlaying	AliyunIEditor.isPlaying
AliyunIPlayer.pause	AliyunIEditor.pause
AliyunIPlayer.resume	AliyunIEditor.resume
AliyunIPlayer.seek	AliyunIEditor.seek
AliyunIPlayer.setAudioSilense	AliyunIEditor.setAudioSilense
AliyunIPlayer.setDisplayMode	AliyunIEditor.setDisplayMode
AliyunIPlayer.setFillBackgroundColor	AliyunIEditor.setFillBackgroundColor
AliyunIPlayer.setOnPlayCallbackListene（去掉）	AliyunIPlayer.setOnPreparedListener（去掉）
AliyunIPlayer.setVolume	AliyunIEditor.setVolume
AliyunIPlayer.start	AliyunIEditor.start
AliyunIPlayer.stop	AliyunIEditor.stop

说明 该版本去掉了OnPreparedListener这个接口，意味着编辑不需要再等待OnPrepared回调了，只要AliyunIEditor.init成功以后就可以添加特效了。

其他

- 设置混音权重applyMusicMixWeight接口增加了参数id，主要是因为这个版本支持多股配音流，所以需要ID来区分，关于接口的详细描述可以参考接口文档。
- getExporter接口已不存在，相关的合成接口直接使用AliyunIEditor中的即可。

旧接口	对应新接口
AliyunIExporter.startCompose	compose
AliyunIExporter.cance	cancelCompose
AliyunIExporter.setTailWatermark（去掉了）	AliyunIExporter.clearTailWatermark（去掉了）

- AliyunICompose.startCompose参数有所变化，OnComposeCallback变为AliyunIComposeCallBack。
- 创建合成实例前需要调用AliyunIEditor#saveEffectToLocal()。

说明 其他未在文档中标述的接口参数变化，在编译阶段会报错，可以参考接口文档中对于新参数的描述进行修改。

4.2. 集成SDK

本文为您介绍Android短视频SDK的接口文档以及集成操作。

前提条件

开发前的环境要求如下表所示。

前提条件

类别	说明
系统版本	支持Android4.3及以上版本。
Java版本	支持Java1.7及以上版本。
API LEVEL	支持ANDROID SDK API LEVEL 18及以上版本。
Android Studio版本	支持Android Studio 2.3以上版本，下载Android Studio。

背景信息

短视频SDK分为专业版、标准版和基础版，各版本之间存在差异。更多信息，请参见短视频SDK产品介绍。

Maven方式集成（推荐）

1. 添加阿里云Maven仓库。

   *在项目级的build.gradle文件*中添加Maven仓库地址。

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

2. 在app工程目录下的*build.gradle*文件中加入对应依赖项，如下表所示。

   > 注意
   >
   > 如果使用短视频SDK 3.24.0及以上版本，请务必使用FFmpeg 4.3.0及以上版本。

版式	Maven项目中依赖项

版式	Maven项目中依赖项

专业版	dependencies{
	implementation 'com.aliyun.video.android:svideopro:3.32.0'//专业版短视频SDK必须依赖。
	implementation 'com.aliyun.video.android:core:1.2.2' //核心库必须依赖。
	implementation 'com.alivc.conan:AlivcConan:1.0.3'//核心库必须依赖。
	//AlivcFFmpeg必须依赖,且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2.0-part' //短视频SDK专用，包size较小。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2' //短视频SDK与播放器SDK共用。两个SDK同时接入时，请用该版本。
	implementation 'com.aliyun.video.android:upload:1.6.6' //上传库，如不需要上传可不依赖。
	implementation 'com.google.code.gson:gson:2.8.0'//第三方依赖库。
	implementation 'com.squareup.okhttp3:okhttp:3.2.0'//第三方依赖库。
	implementation 'com.aliyun.dpa:oss-android-sdk:+'//短视频上传需要依赖上传SDK及OSS，如果不需要上传可不依赖。
	}

标准版	dependencies {
	implementation 'com.aliyun.video.android:svideostandard:3.32.0'//标准版短视频SDK必须依赖。
	implementation 'com.aliyun.video.android:core:1.2.2' //核心库必须依赖。
	implementation 'com.alivc.conan:AlivcConan:1.0.3'//核心库必须依赖。
	//AlivcFFmpeg必须依赖,且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2.0-part' // 短视频SDK专用，包size较小。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2' //短视频SDK与播放器SDK共用。两个SDK同时接入时，请用该版本。
	implementation 'com.aliyun.video.android:upload:1.6.6'// 上传库，如不需要上传可不依赖。
	implementation 'com.google.code.gson:gson:2.8.0'//第三方依赖库。
	implementation 'com.squareup.okhttp3:okhttp:3.2.0'//第三方依赖库。
	implementation 'com.aliyun.dpa:oss-android-sdk:+'//短视频上传需要依赖上传SDK及OSS，如果不需要上传可不依赖。
	}

版式	Maven项目中依赖项

基础版	dependencies {
	implementation 'com.aliyun.video.android:svideosnap:3.32.0'//基础版短视频SDK必须依赖。
	implementation 'com.aliyun.video.android:core:1.2.2' //核心库必须依赖。
	implementation 'com.alivc.conan:AlivcConan:1.0.3'//核心库必须依赖。
	//AlivcFFmpeg必须依赖,且版本需要在4.3.0及以上，在以下两个版本中选择一个版本进行依赖。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2.0-part' // 短视频SDK专用，包size较小。
	implementation 'com.aliyun.video.android:AlivcFFmpeg:4.3.2' //短视频SDK与播放器SDK共用。两个SDK同时接入时，请用该版本。
	implementation 'com.aliyun.video.android:upload:1.6.6'// 上传库，如不需要上传可不依赖。
	implementation 'com.google.code.gson:gson:2.8.0'//第三方依赖库。
	implementation 'com.squareup.okhttp3:okhttp:3.2.0'//第三方依赖库。
	implementation 'com.aliyun.dpa:oss-android-sdk:+'//短视频上传需要依赖上传SDK及OSS，如果不需要上传可不依赖。
	}

说明

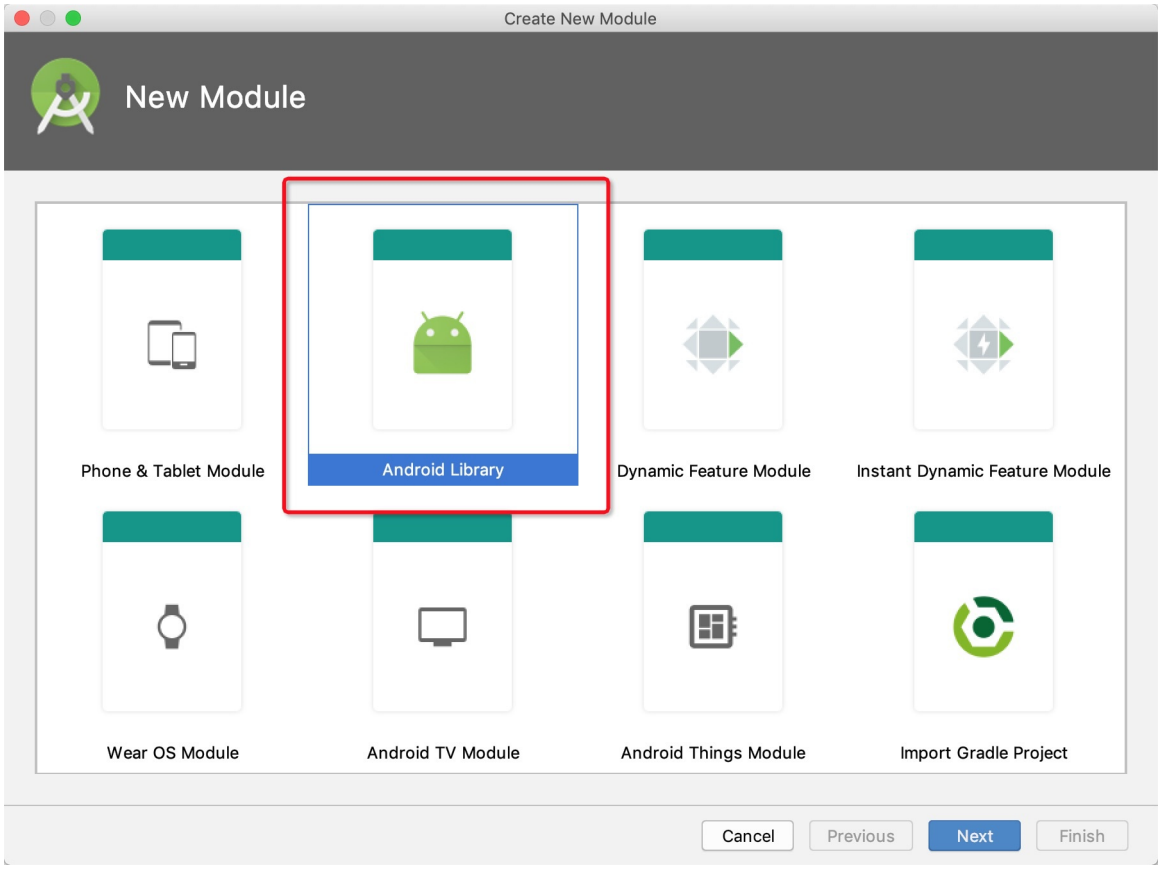
目前短视频SDK仅包含armeabi-v7a和arm64-v8a指令集的SDK，如需要armeabi的指令集的so建议直接通过armeabi-v7a的so拷贝至armeabi以此兼容。需要说明的是考虑到Android的发展和短视频SDK仅支持4.3以上，所以ARMv5、ARMv6设备兼容意义不大。

手动方式集成

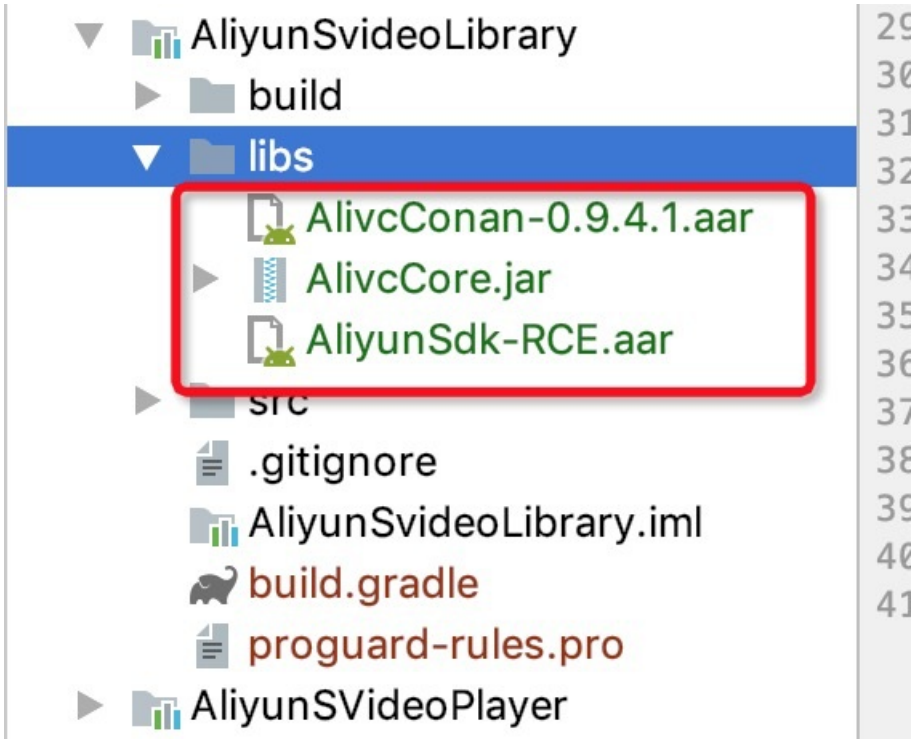
导入aar

1. 创建Module。

   在**Android Studio**中选择**File > New > Newmodule**，新建Library形式的Module。示例名：AliyunSvideoLibrary。



2. 将下载的SDK目录中的*AliyunSdk-RCE.aar*、*AlivcConan-x.x.x.aar*、*AlivcCore.jar*拷贝至新建的Module以module的方式添加到工程中。



3. 通过gradle添加AAR和jar包依赖。

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

4. 在app module中添加AliyunSvideoLibrary Module的依赖。

```
 dependencies {
    implementation project(":AliyunSvideoLibrary")
}
```

导入.so文件

1. 将SDK解压包中的jniLibs目录拷贝到app module的main目录中，并且在app module的build.gradle文件中声明jniLibs的路径。

   示例代码如下所示。

```
android {
 sourceSets.main {
     jni.srcDirs = []
     jniLibs.srcDir "src/main/jniLibs"
 }
}
```

2. SDK加载的动态链接库。

   在最新的版本中已经在SDK内部加载，不需要用户手动加载。

   库文件说明如下所示。

```
libalivcffmpeg.so-------------SDK依赖的第三方库（必须load）
libaliresample.so--------------------音频重采样相关的库（不需要可以去掉）
```

3. 添加上传依赖项。

   短视频上传需要依赖上传SDK及OSS，如果不需要可不添加。

```
implementation 'com.aliyun.video.android:upload:1.6.4'
implementation 'com.google.code.gson:gson:2.8.0'
implementation 'com.squareup.okhttp3:okhttp:3.2.0'
implementation 'com.aliyun.dpa:oss-android-sdk:+'
```

4. 初始化相关配置。

   在App的Application类onCreate方法中调用，示例代码如下所示。

```
com.aliyun.vod.common.httpfinal.QupaiHttpFinal.getInstance().initOkHttpFinal();
```

配置权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.RECORD_VIDEO" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

> **说明**
>
> Android 6.0以上系统需要做动态权限请求。

配置License

获取到License后，需要按以下操作配置License文件。License的获取及详细信息请参见获取短视频SDK License。

在AndroidManifest.xml文件中添加两个meta-data，第一个key的name为**com.aliyun.alivc_license.licensekey**（全小写），value为LicenseKey的值；第二个key的name为**com.aliyun.alivc_license.licensefile**（全小写），value为内置证书文件的路径。示例代码如下所示：

```
<application
   android:icon="@drawable/icon"
   android:label="@string/app_name" >
    <meta-data
         android:name="com.aliyun.alivc_license.licensekey"
         android:value="yourLicenseKey"/> //请输入您的LicenseKey的值
    <meta-data
       android:name="com.aliyun.alivc_license.licensefile"
       android:value="yourLicenseFile"/> //请输入您的内置证书文件的路径，取值示例：assets/alivc_
license/AliVideoCert.crt
  ……
</application>
```

混淆配置

可以在*proguard-rules.pro*文件中进行混淆配置。示例代码如下所示：

```
######################短视频混淆配置#########################
-keep class com.aliyun.**{*;}
-keep class com.duanqu.**{*;}
-keep class com.qu.**{*;}
-keep class com.alibaba.**{*;}
-keep class component.alivc.**{*;}
-keep class com.alivc.**{*;}
-keep class org.webrtc.**{*;}
-keep class org.ugsv.**{*;}
```

相关文档

API接口信息，请参见API参考。

后续步骤，请参见初始化SDK。

4.3. 初始化SDK

短视频SDK的核心类为AlivcSdkCore，它是短视频SDK功能的总入口，并提供了相应的调试开关，供开发者使用。

前提条件

初始化SDK前需要完成以下操作：

- 获取并开通短视频SDK License，详细内容请参见获取短视频SDK License。
- 集成短视频SDK。从3.29.0版本开始，注册SDK之前，需要先配置License，详细内容请参见集成SDK。

初始化

注意：使用短视频SDK前，必须进行如下初始化操作。接口参数请参考AlivcSdkCore。

```
//初始化
bool result = AlivcSdkCore.register(Context context)
//如果reuslt返回false，表示License初始化失败，请检查License的配置是否正确。
```

> 注意
>
> 如果License处于失效、无效或过期的状态，短视频SDK将不可以再使用。

日志

短视频SDK提供了多种日志级别模式AlivcLogLevel，支持输出日志到文件。打开日志后可通过tag：AliYunLog过滤相应日志信息。

```
//打开日志功能
AlivcSdkCore.setLogLevel(AlivcLogLevel level)

//设置输出日志的文件路径
AlivcSdkCore.setLogPath(String path)

//设置输出日志的级别
AlivcSdkCore.setDebugLoggerLevel(AlivcDebugLoggerLevel level)
```

4.4. 视频录制

4.4.1. 概述

本文介绍视频录制的类型及版本差异。

录制类型

视频录制分为三种类型：基础录制、视频合拍、多源录制。

- 基础录制：摄像头采集录制功能。
- 视频合拍：使用一个已有视频作为样本视频，与摄像头采集的数据按照特定的布局方式进行合拍录制。
- 多源录制：支持View录制、摄像头录制等多种视频采集源按需组合的合拍录制。

版本支持

各版本支持的录制类型如下表所示，表格中的✓表示支持，×表示不支持。

录制类型	基础版	标准版	专业版
基础录制	✓	✓	✓
视频合拍	×	✓	✓
多源录制	×	×	✓

4.4.2. 基础录制

短视频SDK提供基础视频录制，同时支持添加配乐，变速录制，人脸贴纸等录制效果。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

类名	功能
AliyunIRecorder	录制功能核心类，包括录制、设置预览、设置特效、设置回调等核心录制功能。
AliyunRecorderCreator	工厂类，用于获取录制实例。
MediaInfo	录制参数配置类，包括设置视频的宽高、编码器类型、录制采集帧率等录制参数。
RecordCallback	录制回调接口，包括设置录制完成回调、录制进度回调及录制错误回调等。
OnFrameCallBack	相机采集数据回调接口，包括选择预览分辨率的回调、采集帧回调及摄像头开启失败的回调。
OnAudioCallBack	音频回调接口，设置音频pcm数据回调。
AliyunIClipManager	视频录制片段管理接口，包括删除片段、设置录制时长等。

录制流程

> **说明** 录制功能需要获取摄像头和麦克风权限，否则无法录制。

阶段	流程	说明	代码示例
基础	1	创建及销毁录制接口，并配置录制参数。	初始化及参数配置
	2	回调设置。	回调设置
	3	创建开始预览和结束预览。	开启预览
	4	创建开始录制片段、取消录制片段、停止录制片段。	开始录制
	5	创建结束录制相关信息。	结束录制
进阶	6	配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。	录制控制及管理
	7	配置美颜、滤镜、背景音乐等录制特效，可按需配置。	设置特效
	8	设置拍照或人脸识别。	其他功能

初始化及参数配置

初始化AliyunIRecorder类，创建录制接口，并配置录制参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

初始化

```
//创建录制接口
AliyunIRecorder recorder = AliyunRecorderCreator.getRecorderInstance(context);//参数context为当前页面的上下文
//销毁录制接口
//不再使用SDK或者程序退出前销毁录制接口，请务必保证不要中途销毁
recorder.destroy();
```

配置录制参数

```
//设置录制视频的质量
recorder.setVideoQuality(quality);
//设置录制视频的码率
recorder.setVideoBitrate(int bitrate);//单位：Kbps
//设置录制视频输出参数
recorder.setMediaInfo(mediaInfo);//相关参数描述请参考MediaInfo的接口文档
//设置录制视频的输出路径
recorder.setOutputPath(string path);
//设置录制视频的输出GOP
recorder.setGop(int gop);//单位：帧数
```

回调设置

通过设置回调，及时获取音视频处理的进展和状态。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置录制回调
recorder.setRecordCallBack(RecordCallback callBack);
//设置视频帧采集回调
recorder.setOnFrameCallback(OnFrameCallBack callback);
//设置音频采集数据的回调
recorder.setOnAudioCallback(OnAudioCallBack callback);
```

开启预览

预览时，需要设置相对应的SurfaceView容器，同时，在Activity/Fragment的onResume()时进行startPreview，在onPause()进行stopPreview。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置预览View
recorder.setDisplayView(SurfaceView displayView);
//开始预览
recorder.startPreview();
//结束预览
//在Activity/Fragment的onPause()时进行stopPreview
recorder.stopPreview();
```

开始录制

在实际录制过程中，常常不能一次就能录制完成所需视频，而总是不断的停止、取消和重新录制。停止录制时会生成一个视频片段，而取消录制则不会保留当前录制的视频片段。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

开始录制

```
//开始录制
recorder.startRecording();
```

录制片段

```
//开始录制
recorder.startRecording();
//停止录制，生成一个视频片段
recorder.stopRecording();
recorder.startRecording();
// 取消录制，当前的视频片段不会保存
recorder.cancelRecording();
//继续录制下一个视频片段
recorder.startRecording();
recorder.stopRecording();
```

结束录制

结束录制时可生成一个片段拼接的视频或只生成片段视频的配置信息。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//结束录制，并且将录制片段视频拼接成一个视频
recorder.finishRecording();
//结束录制，生成片段视频的配置信息（不拼接片段）
recorder.finishRecordingForEdit();
```

录制控制及管理

配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

录制摄像头控制

```
//获取摄像头数量
AliyunIRrecorder.getCameraCount();
//设置摄像头类型
AliyunIRecorder.setCamera(cameraType);
//设置静音录制
AliyunIRecorder.setMute(boolean isMute);
//设置传感器角度值
//（非常重要，建议仔细阅读接口文档）
AliyunIRecorder.setRotation(int rotation);
//设置录制视频的角度
//（非常重要，建议仔细阅读接口文档）
AliyunIRecorder.setRecordRotation(int rotation);
//设置摄像头预览参数（闪光灯、对焦模式、Zoom、曝光度），也可参考下面的接口单独设置各预览参数
AliyunIRecorder.setCameraParam(CameraParam cameraParam);
//切换摄像头
AliyunIRecorder.switchCamera();
//设置闪光灯模式
AliyunIRecorder.setLight(flashType);
//设置Zoom
AliyunIRecorder.setZoom(float rate);
//设置曝光度
AliyunIRecorder.setExposureCompensationRatio(float value);
//设置对焦模式
AliyunIRecorder.setFocusMode(int mode);
//手动对焦
AliyunIRecorder.setFocus(float xRatio, float yRatio);
```

录制片段管理

```
//获取片段管理器
AliyunIClipManager manager = AliyunIRecorder.getClipManager();
//设置最大录制时长（总录制时长，非单个片段的最大时长）
manager.setMaxDuration(int maxDurationMs);
//设置最小录制时长（总录制时长，非单个片段的最小时长）
manager.setMinDuration(int minDurationMs);
//删除最后一段片段
manager.deletePart();
//删除指定的片段
manager.deletePart(int index);
//删除所有片段
manager.deleteAllPart();
//获取片段总时长
manager.getDuration();
//获取总的片段数量
manager.getPartCount();
//获取片段路径列表
manager.getVideoPathList();
```

设置特效

配置美颜、滤镜、背景音乐等录制特效，可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

滤镜

支持自定义滤镜，滤镜的制作方法请参见滤镜及转场。

```
//设置滤镜
AliyunIRecorder.applyFilter(EffectFilter effectFilter);//参数路径设置为null表示移除滤镜效果
//移除滤镜
AliyunIRecorder.applyFilter(new EffectFilter(null));
```

动效滤镜

```
//设置动效滤镜
AliyunIRecorder.applyAnimationFilter(EffectFilter effectFilter);
//移除动效滤镜
AliyunIRecorder.removeAnimationFilter(EffectFilter effctFilter);
```

背景音乐

```
//设置背景音乐
AliyunIRecorder.setMusic(String path,long startTime,long duration);
//移除背景音乐
AliyunIRecorder.setMusic(null, 0, 0);
```

变速

```
//设置录制速度
AliyunIRecorder.setRate(float rate);
```

静态贴纸/水印

```
//添加静态贴纸/水印
AliyunIRecorder.addImage(EffectImage effctImage);
//移除静态贴纸/水印
AliyunIRecorder.removeImage(EffectImage effctImage);
//更新静态贴纸/水印位置
AliyunIRecorder.setEffectView(float xRatio,float yRatio,float widthRatio,float heightRatio,
EffectBase effectBase);
```

动态贴纸

支持自定义动态贴纸，动态贴纸的制作方法请参见动图。

```
//添加动态贴纸
AliyunIRecorder.addPaster(EffectPaster effectPaster,float sx,float sy,float sw,float sh,flo
at rotation,boolean flip);
//移除动态贴纸
AliyunIRecorder.removePaster(EffectPaster effectPaster);
//更新动态贴纸位置
AliyunIRecorder.setEffectView(float xRatio,float yRatio,float widthRatio,float heightRatio,
EffectBase effectBase);
```

高级美颜

视频录制模块，提供了基础的内置美颜功能，同时也支持使用外置的美颜SDK功能，如阿里云美颜特效SDK、相芯科技（FaceUnity）等美颜SDK。内置美颜功能相对比较简单，仅能设置不同的美颜等级；外置美颜SDK通常提供了更为丰富的美颜、美型、美妆美化、滤镜贴纸等功能。

- 内置美颜

```
//设置美颜开关
AliyunIRecorder.setBeautyStatus(boolean on);
//设置美颜程度
AliyunIRecorder.setBeautyLevel(int level);
```

- 外置美颜SDK

  想要在短视频SDK中使用外置美颜SDK所提供的特效，则需要提前获取相应外置美颜SDK的权限并将外置美颜SDK集成接入到短视频SDK中。

  - 阿里云美颜特效SDK的集成接入等相关操作请参见美颜特效SDK。特效设置代码示例请参见使用示例。
  - FaceUnity的购买、集成、使用等相关操作，请参见FaceUnity。

  外置美颜SDK需要获取两个数据用于实现美颜功能：相机纹理ID和相机原始帧数据。以下代码展示如何获取相机纹理ID和相机原始帧数据。

  - 获取相机纹理ID数据

```
AliyunIRecorder.setOnTextureIdCallback(new OnTextureIdCallBack() {
            @Override
            public int onTextureIdBack(int textureId, int textureWidth, int textureHeig
ht, float[] matrix) {
                if (mBeautyInterface != null) {
                    return mBeautyInterface.onTextureIdBack(textureId, textureWidth, te
xtureHeight, matrix, mControlView.getCameraType().getType());
                }
                return textureId;
            }
            @Override
            public int onScaledIdBack(int scaledId, int textureWidth, int textureHeight
, float[] matrix) {
                return scaledId;
            }
            @Override
            public void onTextureDestroyed() {
                //关于自定义渲染（第三方渲染）销毁gl资源，SDK3.7.8之前版本，GLSurfaceView时可以通过GLSurfaceView.queueEvent来销毁gl资源；SDK3.7.8及之后版本开始，推荐在此回调中统一销毁gl资源
                if (mBeautyInterface != null) {
                    mBeautyInterface.release();
                    mBeautyInterface = null;
                }
            }
        });
```

- 获取相机原始帧数据

```
AliyunIRecorder.setOnFrameCallback(new OnFrameCallBack() {
            @Override
            public void onFrameBack(byte[] bytes, int width, int height, Camera.CameraI
nfo info) {
                //原始数据回调NV21，此处获取原始数据主要是提供给FaceUnity高级美颜使用
                if (mBeautyInterface != null) {
                    mBeautyInterface.onFrameBack(bytes, width, height, info);
                }
            }
            @Override
            public Camera.Size onChoosePreviewSize(List<Camera.Size> supportedPreviewSi
zes,
                                                   Camera.Size preferredPreviewSizeForV
ideo) {
                return null;
            }
            @Override
            public void openFailed() {
            }
        });
```

其他功能

支持录制视频时拍照及人脸识别。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

拍照

拍照分为带特效拍照及系统拍照（不带特效），拍照后会通过 `RecordCallback.onPictureBack(Bitmap)` 或者 `RecordCallback.onPictureDataBack(byte[])` 返回数据。

```
//带特效拍照
AliyunIRecorder.takePhoto(boolean needBitmap);
//系统拍照（不带特效）
AliyunIRecorder.takePicture(boolean needBitmap);
//设置系统拍照的照片大小（仅系统拍照支持，带特效拍照不支持）
AliyunIRecorder.setPictureSize(Camera.Size size);
```

人脸识别

人脸识别时，需要接入APP自行内置的人脸识别模型文件，可参考人脸识别模型文件Demo来设置。

```
//开启人脸识别
AliyunIRecorder.needFaceTrackInternal(boolean need);
//设置人脸识别模型文件
AliyunIRecorder.setFaceTrackInternalModelPath(String path);
//设置人脸识别角度
//（非常重要，建议仔细阅读接口文档）
AliyunIRecorder.setFaceDetectRotation(int rotation);
//设置人脸识别数量
//设置内置人脸识别的最大识别数，最大为3个
AliyunIRecorder.setFaceTrackInternalMaxFaceCount(int maxFaceCount);
//添加人脸动图
AliyunIRecorder.addPaster(EffectPaster effectPaster);
```

基础录制代码示例

```
import android.graphics.Bitmap;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceView;
import android.widget.ImageView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.aliyun.svideosdk.common.struct.recorder.MediaInfo;
import com.aliyun.svideosdk.recorder.AliyunIRecorder;
import com.aliyun.svideosdk.recorder.RecordCallback;
import com.aliyun.svideosdk.recorder.impl.AliyunRecorderCreator;
import com.svideo.guide.R;
import java.io.FileOutputStream;
/**
 * 视频录制Example
 */
class RecordActivity : AppCompatActivity() {
    enum class RecordStatus {
        Idle,
        Recording
    }
    private lateinit var mAliyunRecord : AliyunIRecorder
    private lateinit var mCameraViiew : SurfaceView
    private lateinit var mRecordBtn : ImageView
    private var mRecordStatus = RecordStatus.Idle
    companion object {
        const val TAG = "RecordActivity"
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_record)
        mCameraViiew = findViewById(R.id.cameraPreviewView)
        mRecordBtn = findViewById(R.id.btnRecordControl)
        //录制按钮
        mRecordBtn.setOnClickListener {
            if(mRecordStatus == RecordStatus.Recording) {
                mAliyunRecord.stopRecording()
                Toast.makeText(this@RecordActivity, "结束录制", Toast.LENGTH_SHORT).show()
```

```
                updateRecordStatus(RecordStatus.Idle)
            } else {
                mAliyunRecord.takePhoto(true)
            }
        }
        mRecordBtn.setOnLongClickListener {
            if(mRecordStatus == RecordStatus.Idle) {
                val curTime = System.currentTimeMillis()
                mAliyunRecord.setOutputPath("/storage/emulated/0/DCIM/Camera/svideo_record_
video_$curTime.mp4")
                mAliyunRecord.startRecording()
                Toast.makeText(this@RecordActivity, "开始录制", Toast.LENGTH_SHORT).show()
                updateRecordStatus(RecordStatus.Recording)
            }
            true
        }
        mAliyunRecord = AliyunRecorderCreator.getRecorderInstance(this)
        val mediaInfo = MediaInfo()
        mediaInfo.fps = 30
        mediaInfo.crf = 6
        mediaInfo.videoWidth = 720
        mediaInfo.videoHeight = 1080
        mAliyunRecord.setMediaInfo(mediaInfo)
        mAliyunRecord.setDisplayView(mCameraViiew)
        mAliyunRecord.setRecordCallBack(object : RecordCallback {
            override fun onComplete(validClip: Boolean, clipDuration: Long) {
                Log.i(TAG, "onComplete")
            }
            override fun onFinish(outputPath: String?) {
                Log.i(TAG, "onFinish path : $outputPath")
            }
            override fun onProgress(duration: Long) {
            }
            override fun onMaxDuration() {
                Log.i(TAG, "onMaxDuration")
            }
            override fun onError(errorCode: Int) {
                Log.i(TAG, "onError : $errorCode")
            }
            override fun onInitReady() {
                Log.i(TAG, "onInitReady")
            }
            override fun onDrawReady() {
                Log.i(TAG, "onDrawReady")
            }
            override fun onPictureBack(bitmap: Bitmap?) {
                Log.i(TAG, "onPictureBack")
                val curTime = System.currentTimeMillis()
                val outputFile = "/storage/emulated/0/DCIM/Camera/svideo_record_photo_$curT
ime.png"
                val outputStream = FileOutputStream(outputFile)
                bitmap?.compress(Bitmap.CompressFormat.PNG, 100, outputStream)
                outputStream.flush()
                outputStream.close()
```

```
                runOnUiThread {
                    Toast.makeText(this@RecordActivity, "图片已保存到相册", Toast.LENGTH_SHOR
T).show()
                }
            }
            override fun onPictureDataBack(p0: ByteArray?) {
                Log.i(TAG, "onPictureDataBack")
            }
        })
    }
    private fun updateRecordStatus(recordStatus: RecordStatus)
    {
        mRecordStatus = recordStatus
        when(recordStatus) {
            RecordStatus.Idle -> {
                mRecordBtn.setImageResource(R.mipmap.alivc_svideo_bg_record_start)
            }
            RecordStatus.Recording -> {
                mRecordBtn.setImageResource(R.mipmap.alivc_svideo_bg_record_storp)
            }
        }
    }
    override fun onResume() {
        super.onResume()
        mAliyunRecord.startPreview()
    }
    override fun onPause() {
        super.onPause()
        mAliyunRecord.stopPreview()
    }
    override fun onDestroy() {
        super.onDestroy()
        mAliyunRecord.destroy()
    }
}
```

XML文件配置示例

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.co
m/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"
    tools:context=".MainActivity">
    <SurfaceView
        android:id="@+id/cameraPreviewView"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </SurfaceView>
    <ImageView
        android:id="@+id/btnRecordControl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="50dp"
        android:src="@mipmap/alivc_svideo_bg_record_storp"
        android:visibility="visible">
    </ImageView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

4.4.3. 视频合拍

短视频SDK提供视频合拍功能AliyunIMixRecorder。使用一个已有视频作为样本视频，与摄像头采集的数据按照特定的布局方式（例如左右分屏，上下分屏，画中画等）进行合拍录制。视频合拍是基础录制的功能升级，相比基础录制，视频合拍增加了一个新的本地视频轨道。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

概念介绍

在以下文档介绍中将提及一些特殊概念，为方便开发者理解，可预先对视频合拍、轨道及轨道布局的概念做相关了解。

相关类功能

类名	功能
AliyunIMixRecorder	录制功能核心类，包括录制、设置预览、设置特效、设置回调等视频合拍的核心录制功能。
AliyunMixRecorderCreator	工厂类，用于创建合拍录制实例。
AliyunMixMediaInfoParam	合拍参数配置类，包括设置合拍视频的轨道排列及输出路径等参数。
AliyunMixRecorderDisplayParam	合拍轨道配置类，包括设置轨道的布局信息及布局层级等参数。
MediaInfo	录制参数配置类，包括设置视频的宽高、编码器类型、录制采集帧率等录制参数。
RecordCallback	录制回调接口，包括设置录制完成回调、录制进度回调及录制错误回调等。
OnFrameCallBack	相机采集数据回调接口，包括选择预览分辨率的回调、采集帧回调及摄像头开启失败的回调。
OnAudioCallBack	音频回调接口，设置音频pcm数据回调。
AliyunIClipManager	视频录制片段管理接口，包括删除片段、设置录制时长等。

合拍流程

> **说明** 合拍录制功能需要获取摄像头和麦克风权限，否则无法录制。

视频合拍流程与基础录制流程基本相同，主要差别在于配置录制参数时设置输入输出参数以及设置预览View上的差别。

阶段	流程	说明	示例代码
基础	1	创建及销毁录制接口，并配置录制参数。	初始化及参数配置
	2	回调设置。	回调设置
	3	设置预览View并开启预览。	开启预览
	4	创建开始录制片段、取消录制片段、停止录制片段。	开始录制
	5	创建结束录制相关信息。	结束录制

阶段	流程	说明	示例代码
进阶	6	配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。	录制控制及管理
	7	配置美颜、滤镜等录制特效，可按需配置。	设置特效
	8	设置拍照或人脸识别。	其他功能

初始化及参数配置

初始化AliyunIMixRecorder类，创建录制接口，并配置录制参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

初始化

```
//创建录制接口
AliyunIMixRecorder recorder = AliyunMixRecorderCreator.createAlivcMixRecorderInstance(conte
xt);
//销毁录制接口
//不再使用SDK或者程序退出前销毁录制接口，请务必保证不要中途销毁
AliyunIMixRecorder.release();
```

配置录制参数

```
//设置录制视频的质量
AliyunIMixRecorder.setVideoQuality(quality);
//设置录制视频的码率
AliyunIMixRecorder.setVideoBitrate(int bitrate);//单位：Kbps
//设置录制视频输出参数
//inputMediaInfo为合成视频的参数(轨道排列及输出视频)，outputInfo为输出视频的信息
AliyunIMixRecorder.setMixMediaInfo(AliyunMixMediaInfoParam inputMediaInfo, MediaInfo output
Info);//相关参数描述请参考AliyunMixMediaInfoParam和MediaInfo接口文档
//设置录制视频的输出路径
AliyunIMixRecorder.setOutputPath(String path);
//设置录制视频的输出GOP
AliyunIMixRecorder.setGop(int gop);//单位：帧数
```

回调设置

通过设置回调，及时获取音视频处理的进展和状态。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置录制回调
AliyunIMixRecorder.setRecordCallBack(RecordCallback callBack);
//设置视频帧采集回调
AliyunIMixRecorder.setOnFrameCallback(OnFrameCallBack callback);
//设置音频采集数据的回调
AliyunIMixRecorder.setOnAudioCallback(OnAudioCallBack callback);
```

开启预览

预览时，需要设置相对应的SurfaceView容器，同时，在Activity/Fragment的onResume()时进行startPreview，在onPause()进行stopPreview。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置预览View
//cameraView参数是相机的视图，videoView参数是本地视频的视图
AliyunIMixRecorder.setDisplayView(SurfaceView cameraView, Surface videoView);
//开始预览
AliyunIMixRecorder.startPreview();
//结束预览
//在Activity/Fragment的onPause()时进行stopPreview
AliyunIMixRecorder.stopPreview();
```

开始录制

在实际录制过程中，常常不能一次性就能录制完成所需视频，而总是不断的停止、取消和重新录制。停止录制时会生成一个视频片段，而取消录制则不会保留当前录制的视频片段。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

开始录制

```
//开始录制
AliyunIMixRecorder.startRecording();
```

录制片段

```
//开始录制
AliyunIMixRecorder.startRecording();
//停止录制，生成一个视频片段
AliyunIMixRecorder.stopRecording();
AliyunIMixRecorder.startRecording();
// 取消录制，当前的视频片段不会保存
AliyunIMixRecorder.cancelRecording();
//继续录制下一个视频片段
AliyunIMixRecorder.startRecording();
AliyunIMixRecorder.stopRecording();
```

结束录制

结束录制时可生成一个片段拼接的视频或只生成片段视频的配置信息。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//结束录制，并且将录制片段视频拼接成一个视频
AliyunIMixRecorder.finishRecording();
//结束录制，生成片段视频的配置信息（不拼接片段）
AliyunIMixRecorder.finishRecordingForEdit();
```

录制控制及管理

该模块用于配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

录制摄像头控制

```
//获取摄像头数量
AliyunIMixRecorder.getCameraCount();
//设置摄像头类型
AliyunIMixRecorder.setCamera(cameraType);
//设置静音录制
AliyunIMixRecorder.setMute(boolean isMute);
//设置传感器角度值
//（非常重要，建议仔细阅读接口文档）
AliyunIMixRecorder.setRotation(int rotation);
//设置录制视频的角度
//（非常重要，建议仔细阅读接口文档）
AliyunIMixRecorder.setRecordRotation(int rotation);
//设置摄像头预览参数（闪光灯、对焦模式、Zoom、曝光度），也可参考下面的接口单独设置各预览参数
AliyunIMixRecorder.setCameraParam(CameraParam cameraParam);
//切换摄像头
AliyunIMixRecorder.switchCamera();
//设置闪光灯模式
AliyunIMixRecorder.setLight(FlashType flashType);
//设置Zoom
AliyunIMixRecorder.setZoom(float rate);
//设置曝光度
AliyunIMixRecorder.setExposureCompensationRatio(float value);
//设置对焦模式
AliyunIMixRecorder.setFocusMode(int mode);
//手动对焦
AliyunIMixRecorder.setFocus(float xRatio, float yRatio);
```

录制片段管理

```
//获取片段管理器
AliyunIClipManager manager = AliyunIRecorder.getClipManager();
//设置最大录制时长（总录制时长，非单个片段的最大时长）
manager.setMaxDuration(int maxDurationMs);
//设置最小录制时长（总录制时长，非单个片段的最小时长）
manager.setMinDuration(int minDurationMs);
//删除最后一段片段
manager.deletePart();
//删除指定的片段
manager.deletePart(int index);
//删除所有片段
manager.deleteAllPart();
//获取片段总时长
manager.getDuration();
//获取总的片段数量
manager.getPartCount();
//获取片段路径列表
manager.getVideoPathList();
```

设置特效

该模块用于配置美颜、滤镜等录制特效，可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

滤镜

支持自定义滤镜，滤镜的制作方法请参见滤镜及转场。

```
//设置滤镜
AliyunIMixRecorder.applyFilter(effectFilter);
//移除滤镜
//参数路径设置为null表示移除滤镜效果
AliyunIMixRecorder.applyFilter(new EffectFilter(null));
```

动效滤镜

```
//设置动效滤镜
AliyunIMixRecorder.applyAnimationFilter(effectFilter);
//移除动效滤镜
AliyunIMixRecorder.removeAnimationFilter(effctFilter);
```

变速

```
//设置录制速度
AliyunIMixRecorder.setRate(float rate);
```

静态贴纸/水印

```
//添加静态贴纸/水印
AliyunIMixRecorder.addImage(effctImage);
//移除静态贴纸/水印
AliyunIMixRecorder.removeImage(effctImage);
//更新静态贴纸/水印位置
AliyunIMixRecorder.setEffectView(float xRatio,float yRatio,float widthRatio,float heightRat
io,EffectBase effectBase);
```

动态贴纸

支持自定义动态贴纸，动态贴纸的制作方法请参见动图。

```
//添加动态贴纸
AliyunIMixRecorder.addPaster(effectPaster,float sx,float sy,float sw,float sh,float rotatio
n,boolean flip);
//移除动态贴纸
AliyunIMixRecorder.removePaster(effectPaster);
//更新动态贴纸位置
AliyunIMixRecorder.setEffectView(float xRatio,float yRatio,float widthRatio,float heightRat
io,effectBase);
```

高级美颜

视频录制模块，提供了基础的内置美颜功能，同时也支持使用外置的美颜SDK功能，如阿里云美颜特效SDK、相芯科技（FaceUnity）等美颜SDK。内置美颜功能相对比较简单，仅能设置不同的美颜等级；外置美颜SDK通常提供了更为丰富的美颜、美型、美妆美化、滤镜贴纸等功能。

- 内置美颜

  ```
  //设置美颜开关
  AliyunIMixRecorder.setBeautyStatus(boolean on);
  //设置美颜程度
  AliyunIMixRecorder.setBeautyLevel(int level);
  ```

- 外置美颜SDK

  想要在短视频SDK中使用外置美颜SDK所提供的特效，则需要提前获取相应外置美颜SDK的权限并将外置美颜SDK集成接入到短视频SDK中。

  - 阿里云美颜特效SDK的集成接入等相关操作请参见美颜特效SDK。特效设置代码示例请参见使用示例。
  - FaceUnity的购买、集成、使用等相关操作，请参见FaceUnity。

  外置美颜SDK需要获取两个数据用于实现美颜功能：相机纹理ID和相机原始帧数据。以下代码展示如何获取相机纹理ID和相机原始帧数据。

- 获取相机纹理ID数据

```
AliyunIMixRecorder.setOnTextureIdCallback(new OnTextureIdCallBack() {
            @Override
            public int onTextureIdBack(int textureId, int textureWidth, int textureHeig
ht, float[] matrix) {
                if (mBeautyInterface != null) {
                    return mBeautyInterface.onTextureIdBack(textureId, textureWidth, te
xtureHeight, matrix, mControlView.getCameraType().getType());
                }
                return textureId;
            }
            @Override
            public int onScaledIdBack(int scaledId, int textureWidth, int textureHeight
, float[] matrix) {
                return scaledId;
            }
            @Override
            public void onTextureDestroyed() {
                //关于自定义渲染（第三方渲染）销毁gl资源，SDK3.7.8之前版本，GLSurfaceView时可以通过GLSurfaceView.queueEvent来销毁gl资源；SDK3.7.8及之后版本开始，推荐在此回调中统一销毁gl资源
                if (mBeautyInterface != null) {
                    mBeautyInterface.release();
                    mBeautyInterface = null;
                }
            }
        });
```

- 获取相机原始帧数据

```
AliyunIMixRecorder.setOnFrameCallback(new OnFrameCallBack() {
            @Override
            public void onFrameBack(byte[] bytes, int width, int height, Camera.CameraI
nfo info) {
                //原始数据回调NV21,此处获取原始数据主要是提供给FaceUnity高级美颜使用
                if (mBeautyInterface != null) {
                    mBeautyInterface.onFrameBack(bytes, width, height, info);
                }
            }
            @Override
            public Camera.Size onChoosePreviewSize(List<Camera.Size> supportedPreviewSi
zes,
                                                   Camera.Size preferredPreviewSizeForV
ideo) {
                return null;
            }
            @Override
            public void openFailed() {
            }
        });
```

其他功能

支持录制视频时拍照及人脸识别。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

拍照

拍照分为带特效拍照及系统拍照（不带特效），拍照后会通过RecordCallback.onPictureBack(Bitmap)或者RecordCallback.onPictureDataBack(byte[]) 返回数据。

```
//带特效拍照
AliyunIMixRecorder.takePhoto(boolean needBitmap);
//系统拍照（不带特效）
AliyunIMixRecorder.takePicture(boolean needBitmap);
//设置系统拍照的照片大小（仅系统拍照支持，带特效拍照不支持）
AliyunIMixRecorder.setPictureSize(Camera.Size size);
```

人脸识别

人脸识别时，需要接入APP自行内置的人脸识别模型文件，可参考人脸识别模型文件Demo来设置。

```
//开启人脸识别
AliyunIMixRecorder.needFaceTrackInternal(boolean need);
//设置人脸识别模型文件
AliyunIMixRecorder.setFaceTrackInternalModelPath(String path);
//设置人脸识别角度
//（非常重要，建议仔细阅读接口文档）
AliyunIMixRecorder.setFaceDetectRotation(int rotation);
//设置人脸识别数量
//设置内置人脸识别的最大识别数，最大为3个
AliyunIMixRecorder.setFaceTrackInternalMaxFaceCount(int maxFaceCount);
//添加人脸动图
AliyunIMixRecorder.addPaster(EffectPaster effectPaster);
```

视频合拍代码示例

```
import android.graphics.Bitmap
import android.os.Bundle
import android.util.Log
import android.view.SurfaceView
import android.widget.ImageView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.aliyun.svideosdk.common.struct.common.VideoDisplayMode
import com.aliyun.svideosdk.common.struct.encoder.VideoCodecs
import com.aliyun.svideosdk.common.struct.recorder.MediaInfo
import com.aliyun.svideosdk.mixrecorder.AliyunIMixRecorder
import com.aliyun.svideosdk.mixrecorder.AliyunMixMediaInfoParam
import com.aliyun.svideosdk.mixrecorder.AliyunMixRecorderDisplayParam
import com.aliyun.svideosdk.mixrecorder.AliyunMixTrackLayoutParam
import com.aliyun.svideosdk.mixrecorder.impl.AliyunMixRecorderCreator
import com.aliyun.svideosdk.recorder.RecordCallback
/**
 * 视频录制Example
 */
class MixRecordActivity : AppCompatActivity() {
```

```
    enum class RecordStatus {
        Idle,
        Recording
    }
    private lateinit var mAliyunRecord : AliyunIMixRecorder
    private lateinit var mVideoPreviewView : SurfaceView
    private lateinit var mCameraPreviewView : SurfaceView
    private lateinit var mRecordBtn : ImageView
    private var mRecordStatus = RecordStatus.Idle
    companion object {
        const val TAG = "MixRecordActivity"
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mixrecord)
        mVideoPreviewView = findViewById(R.id.videoPreviewView)
        mCameraPreviewView = findViewById(R.id.cameraPreviewView)
        mRecordBtn = findViewById(R.id.btnRecordControl)
        //录制按钮
        mRecordBtn.setOnClickListener {
            if(mRecordStatus == RecordStatus.Recording) {
                mAliyunRecord.finishRecording()
                Toast.makeText(this@MixRecordActivity, "结束录制", Toast.LENGTH_SHORT).show(
)
                updateRecordStatus(RecordStatus.Idle)
            } else {
                val curTime = System.currentTimeMillis()
                mAliyunRecord.setOutputPath("/storage/emulated/0/DCIM/Camera/svideo_mixreco
rd_video_$curTime.mp4")
                mAliyunRecord.startRecording()
                Toast.makeText(this@MixRecordActivity, "开始录制", Toast.LENGTH_SHORT).show(
)
                updateRecordStatus(RecordStatus.Recording)
            }
        }
        mAliyunRecord = AliyunMixRecorderCreator.createAlivcMixRecorderInstance(this)
        val videoDisplayParam = AliyunMixRecorderDisplayParam.Builder()
                .displayMode(VideoDisplayMode.FILL)
                .layoutParam(
                        AliyunMixTrackLayoutParam.Builder()
                                .centerX(0.25f)
                                .centerY(0.5f)
                                .widthRatio(0.5f)
                                .heightRatio(1.0f)
                                .build()
                )
                .build()
        val cameraDisplayParam = AliyunMixRecorderDisplayParam.Builder()
                .displayMode(VideoDisplayMode.FILL)
                .layoutParam(AliyunMixTrackLayoutParam.Builder()
                        .centerX(0.75f)
                        .centerY(0.5f)
                        .widthRatio(0.5f)
                        .heightRatio(1.0f)
                        .build())
```

```
                .build()
        val mixMediaParam = AliyunMixMediaInfoParam.Builder()
                .streamStartTimeMills(0L)
                .streamEndTimeMills(0L) //设置为0L就会自动使用视频的时长
                .mixVideoFilePath("/storage/emulated/0/DCIM/Camera/VID_20210317_174802.mp4"
)
                .mixDisplayParam(videoDisplayParam)
                .recordDisplayParam(cameraDisplayParam)
                .build()
        val mediaInfo = MediaInfo()
        mediaInfo.fps = 30
        mediaInfo.crf = 6
        mediaInfo.videoWidth = 720
        mediaInfo.videoHeight = 1080
        mediaInfo.videoCodec = VideoCodecs.H264_SOFT_OPENH264
        mAliyunRecord.setMixMediaInfo(mixMediaParam, mediaInfo)
        mAliyunRecord.setDisplayView(mCameraPreviewView, mVideoPreviewView)
        mAliyunRecord.setRecordCallback(object : RecordCallback {
            override fun onComplete(validClip: Boolean, clipDuration: Long) {
                Log.i(TAG, "onComplete")
            }
            override fun onFinish(outputPath: String?) {
                Log.i(TAG, "onFinish path : $outputPath")
                mAliyunRecord.clipManager.deleteAllPart()
            }
            override fun onProgress(progress: Long) {
                Log.i(TAG, "onProgress  : $progress")
            }
            override fun onMaxDuration() {
                Log.i(TAG, "onMaxDuration")
            }
            override fun onError(errorCode: Int) {
                Log.i(TAG, "onError : $errorCode")
            }
            override fun onInitReady() {
                Log.i(TAG, "onInitReady")
            }
            override fun onDrawReady() {
                Log.i(TAG, "onDrawReady")
            }
            override fun onPictureBack(bitmap: Bitmap?) {
                Log.i(TAG, "onPictureBack")
            }
            override fun onPictureDataBack(p0: ByteArray?) {
                Log.i(TAG, "onPictureDataBack")
            }
        })
    }
    private fun updateRecordStatus(recordStatus: RecordStatus)
    {
        mRecordStatus = recordStatus
        when(recordStatus) {
            RecordStatus.Idle -> {
                mRecordBtn.setImageResource(R.mipmap.alivc_svideo_bg_record_start)
```

```
            }
            RecordStatus.Recording -> {
                mRecordBtn.setImageResource(R.mipmap.alivc_svideo_bg_record_storp)
            }
        }
    }
    override fun onResume() {
        super.onResume()
        mAliyunRecord.startPreview()
    }
    override fun onPause() {
        super.onPause()
        mAliyunRecord.stopPreview()
    }
    override fun onDestroy() {
        super.onDestroy()
        mAliyunRecord.release()
    }
}
```

XML文件配置示例

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.co
m/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"
    tools:context=".MainActivity">
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.5" />
    <SurfaceView
        android:id="@+id/videoPreviewView"
        android:layout_width="0dp"
        android:layout_height="400dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/guideline3"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        tools:layout_editor_absoluteY="0dp" />
    <SurfaceView
        android:id="@+id/cameraPreviewView"
        android:layout_width="0dp"
        android:layout_height="400dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="@+id/guideline3"
        app:layout_constraintEnd_toEndOf="parent" />
    <ImageView
        android:id="@+id/btnRecordControl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_marginBottom="50dp"
        android:src="@mipmap/alivc_svideo_bg_record_start"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent">
    </ImageView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

4.4.4. 多源录制

短视频SDK在基础录制功能的基础上全新升级了录制功能，除了满足基础录制所有的录制能力，还新增支持了View录制（即录屏）。短视频SDK支持摄像头录制、View录制等多种视频采集源按需组合的合拍录制。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

概念介绍

在以下文档介绍中将提及一些特殊概念，为方便开发者理解，请预先对多源录制、轨道及轨道布局的概念做相关了解。

相关类功能

类名	功能
AliyunIVideoRecorder	多源录制核心类，包括录制、设置预览、设置特效、设置回调等多源录制的核心功能。
AliyunMultiRecorderCreator	多源录制工厂类，获取视频录制实例。
AliyunVideoRecorderConfig	视频录制配置信息，包括设置视频的宽高、编码器类型、录制采集帧率等录制参数。
AliyunIVideoCapture	视频数据采集配置接口，包括添加摄像头采集器及添加局部录屏采集器接口。
AliyunICameraCapture	摄像头录制接口。
AliyunIViewCapture	View录制接口。
OnVideoRecordListener	录制监听接口，包括设置录制完成回调、录制进度回调及录制错误回调等。
OnAudioCallBack	音频回调接口，设置音频pcm数据回调。
OnFrameCallBack	相机采集数据回调接口，包括选择预览分辨率的回调、采集帧回调及摄像头开启失败的回调。
OnPictureCallback	拍照截屏回调接口。
AliyunIClipManager	视频录制片段管理接口，包括删除片段、设置录制时长等。

多源录制流程

说明 多源录制功能需要获取摄像头和麦克风权限，否则无法录制。

阶段	流程	说明	示例代码

阶段	流程	说明	示例代码
基础	1	创建及销毁录制接口，并配置录制参数。	初始化及参数配置
	2	配置View录制和摄像头录制的相关参数。	采集源配置
	3	回调设置。	回调设置
	4	设置预览View并开启预览。	开启预览
	5	创建开始录制片段、取消录制片段、停止录制片段。	开始录制
	6	创建结束录制相关信息。	结束录制
进阶	7	配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。	录制控制及管理
	8	配置美颜、滤镜、动态贴纸等录制特效，可按需配置。	设置特效
	9	设置背景音乐、背景图片及变速等。	其他功能

初始化及参数配置

初始化AliyunIVideoRecorder类，创建录制接口，并配置录制参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

初始化

```
//创建录制接口
AliyunIVideoRecorder recorder = AliyunMultiRecorderCreator.getVideoRecorderInstance(context, config);
//销毁录制接口
//不再使用SDK或者程序退出前销毁录制接口，请务必保证不要中途销毁，一般在Activity#onDestroy调用
recorder.destroy();
```

配置录制参数

```
AliyunVideoRecorderConfig config = AliyunVideoRecorderConfig.builder()
                //必需设置的参数
                .videoWidth(***)
                .videoHeight(***)
                .outputPath(***)
                //其他非必须设置的参数
                .***()
                .***()
                .build();
```

采集源配置

按需组合多个采集源，采集源添加完成后必需调用 `recorder.prepare()` 以表示采集源添加完成。

摄像头录制

接口参数请参考AliyunICameraCapture。

```
//1.布局配置
AliyunLayoutParam cameraLayoutParam = AliyunLayoutParam.builder()
                .layoutLevel(***)
                .centerX(***)
                .centerY(***)
                .widthRatio(***)
                .heightRatio(***)
                .displayMode(***)
                .build();
//2.添加采集源
AliyunICameraCapture cameraCapture = recorder.getVideoCapture().addCameraCapture(cameraLayo
utParam);
//3.配置采集源
//设置摄像头预览view，必需属性
cameraCapture.setDisplayView(SurfaceView)
//其他设置（按需设置）
cameraCapture.set***();
```

View录制

接口参数请参考AliyunIViewCapture。

```
//1.布局配置
AliyunLayoutParam viewLayoutParam = AliyunLayoutParam.builder()
                .layoutLevel(***)
                .centerX(***)
                .centerY(***)
                .widthRatio(***)
                .heightRatio(***)
                .displayMode(***)
                .build();
//2.添加采集源
//获取目前录制View
View recordView = getRecordView();
AliyunIViewCapture viewCapture = recorder.getVideoCapture().addViewCapture(viewLayoutParam,
recordView);
//3.配置采集源（按需设置）
viewCapture.set***();
```

采集源准备

添加完所有采集源后必需调用以下接口，以表示采集源添加完成。

```
recorder.prepare();
```

回调设置

通过设置回调，及时获取音视频处理的进展和状态。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置录制回调（按需设置）
recorder.setOnRecordListener(OnVideoRecordListener);
//设置音频采集数据的回调（按需设置）
recorder.setOnAudioCallback(OnAudioCallBack);
```

开启预览

通常在Activity#onResume调用startPreview，在Activity#onPause调用stopPreview。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//开始预览
//一般在Activity#onResume调用
AliyunIRecorder.startPreview();
//结束预览
//一般在Activity#onPause调用
AliyunIRecorder.stopPreview();
```

开始录制

在实际录制过程中，常常不能一次性就能录制完成所需视频，而总是不断的停止、取消和重新录制。停止录制时会生成一个视频片段，而取消录制则不会保留当前录制的视频片段。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

开始录制

```
//开始录制
recorder.startRecording();
```

录制片段

```
//开始录制
recorder.startRecording();
//停止录制，生成一个视频片段
recorder.stopRecording();
recorder.startRecording();
// 取消录制，当前的视频片段不会保存
recorder.cancelRecording();
//继续录制下一个视频片段
recorder.startRecording();
recorder.stopRecording();
```

结束录制

结束录制时可生成一个片段拼接的视频或只生成片段视频的配置信息。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//结束录制，并且将录制片段视频拼接成一个视频
recorder.finishRecording();
//结束录制，生成片段视频的配置信息（不拼接片段）
recorder.finishRecordingForEdit();
```

录制控制及管理

配置录制相机控制相关参数（设置摄像头类型、设置闪光灯模式等）及录制片段管理相关参数（配置录制最大或最小时长、删除录制片段、获取录制片段数量等），可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

录制摄像头控制

```
//获取摄像头数量
AliyunICameraCapture.getCameraCount();
//设置摄像头类型：前置摄像头、后置摄像头
AliyunICameraCapture.setCamera(CameraType cameraType);
//设置传感器角度值
//（非常重要，建议仔细阅读接口文档）
AliyunICameraCapture.setRotation(int rotation);
//设置录制视频的角度
//（非常重要，建议仔细阅读接口文档）
AliyunICameraCapture.setRecordRotation(int rotation);
//设置摄像头预览参数（闪光灯、对焦模式、zoom、曝光度），也可参考下面的接口单独设置各预览参数
AliyunICameraCapture.setCameraParam(CameraParam cameraParam);
//切换摄像头
AliyunICameraCapture.switchCamera();
//设置闪光灯模式
AliyunICameraCapture.setLight(FlashType flashType);
//设置Zoom
AliyunICameraCapture.setZoom(float rate);
//设置曝光度
AliyunICameraCapture.setExposureCompensationRatio(float value);
//设置对焦模式
AliyunICameraCapture.setFocusMode(int mode);
//手动对焦
AliyunICameraCapture.setFocus(float xRatio, float yRatio);
```

录制片段管理

```
//获取片段管理器
AliyunIClipManager manager = AliyunIVideoRecorder.getClipManager();
//设置最大录制时长（总录制时长，非单个片段的最大时长）
manager.setMaxDuration(int maxDurationMs);
//设置最小录制时长（总录制时长，非单个片段的最小时长）
manager.setMinDuration(int minDurationMs);
//删除最后一段片段
manager.deletePart();
//删除指定的片段
manager.deletePart(int index);
//删除所有片段
manager.deleteAllPart();
//获取片段总时长
manager.getDuration();
//获取总的片段数量
manager.getPartCount();
//获取片段路径列表
manager.getVideoPathList();
```

设置特效

配置美颜、滤镜、动态贴纸等录制特效，可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

滤镜

支持自定义滤镜，滤镜的制作方法请参见滤镜及转场。

```
//设置滤镜
AliyunICameraCapture.applyFilter(EffectFilter effectFilter);
//移除滤镜
AliyunICameraCapture.removeFilter();
```

动效滤镜

```
//设置动效滤镜
AliyunICameraCapture.applyAnimationFilter(EffectFilter effectFilter);
//移除动效滤镜
AliyunICameraCapture.removeAnimationFilter(EffectFilter effctFilter);
```

静态贴纸

```
//添加静态贴纸
AliyunICameraCapture.addImage(EffectImage effctImage);
//移除静态贴纸
AliyunICameraCapture.removeImage(EffectImage effctImage);
//更新静态贴纸位置
AliyunICameraCapture.setEffectView(float xRatio,float yRatio,float widthRatio,float heightR
atio,EffectBase effectBase);
```

动态贴纸

支持自定义动态贴纸，动态贴纸的制作方法请参见动图。

```
//添加动态贴纸
AliyunICameraCapture.addPaster(EffectPaster effectPaster,float sx,float sy,float sw,float s
h,float rotation,boolean flip);
//移除动态贴纸
AliyunICameraCapture.removePaster(EffectPaster effectPaster);
//更新动态贴纸位置
AliyunICameraCapture.setEffectView(float xRatio,float yRatio,float widthRatio,float heightR
atio,EffectBase effectBase);
```

高级美颜

视频录制模块，提供了基础的内置美颜功能，同时也支持使用外置的美颜SDK功能，如阿里云美颜特效SDK、相芯科技（FaceUnity）等美颜SDK。内置美颜功能相对比较简单，仅能设置不同的美颜等级；外置美颜SDK通常提供了更为丰富的美颜、美型、美妆美化、滤镜贴纸等功能。

- 内置美颜

  ```
  //设置美颜开关
  AliyunICameraCapture.setBeautyStatus(boolean on);
  //设置美颜程度
  AliyunICameraCapture.setBeautyLevel(int level);
  ```

- 外置美颜SDK

  想要在短视频SDK中使用外置美颜SDK所提供的特效，则需要提前获取相应外置美颜SDK的权限并将外置美颜SDK集成接入到短视频SDK中。

- 阿里云美颜特效SDK的集成接入等相关操作请参见美颜特效SDK。特效设置代码示例请参见使用示例。
- FaceUnity的购买、集成、使用等相关操作，请参见FaceUnity。

外置美颜SDK需要获取两个数据用于实现美颜功能：相机纹理ID和相机原始帧数据。以下代码展示如何获取相机纹理ID和相机原始帧数据。

- 获取相机纹理ID数据

```
AliyunICameraCapture.setOnTextureIdCallback(new OnTextureIdCallBack() {
            @Override
            public int onTextureIdBack(int textureId, int textureWidth, int textureHeig
ht, float[] matrix) {
                if (mBeautyInterface != null) {
                    return mBeautyInterface.onTextureIdBack(textureId, textureWidth, te
xtureHeight, matrix, mControlView.getCameraType().getType());
                }
                return textureId;
            }
            @Override
            public int onScaledIdBack(int scaledId, int textureWidth, int textureHeight
, float[] matrix) {
                return scaledId;
            }
            @Override
            public void onTextureDestroyed() {
                //关于自定义渲染（第三方渲染）销毁gl资源，SDK3.7.8之前版本，GLSurfaceView时可
以通过GLSurfaceView.queueEvent来销毁gl资源；SDK3.7.8及之后版本开始，推荐在此回调中统一销毁gl资
源
                if (mBeautyInterface != null) {
                    mBeautyInterface.release();
                    mBeautyInterface = null;
                }
            }
        });
```

- 获取相机原始帧数据

```
AliyunICameraCapture.setOnFrameCallback(new OnFrameCallBack() {
            @Override
            public void onFrameBack(byte[] bytes, int width, int height, Camera.CameraI
nfo info) {
                //原始数据回调 NV211，此处获取原始数据主要是提供给FaceUnity高级美颜使用
                if (mBeautyInterface != null) {
                    mBeautyInterface.onFrameBack(bytes, width, height, info);
                }
            }
            @Override
            public Camera.Size onChoosePreviewSize(List<Camera.Size> supportedPreviewSi
zes,
                                                   Camera.Size preferredPreviewSizeForV
ideo) {
                return null;
            }
            @Override
            public void openFailed() {
            }
        });
```

其他功能

支持添加背景音乐、添加水印、变速等功能。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

变速

```
//设置录制速度
AliyunIVideoRecorder.setRate(float rate);
```

静音

```
//设置麦克风静音
AliyunIVideoRecorder.setMute(boolean isMute);
```

退出自动清除

```
//设置退出录制时，是否自动清空录制的分段视频文件
AliyunIVideoRecorder.setIsAutoClearClipVideos(boolean isAutoClear);
```

背景音乐

```
//设置背景音乐
AliyunIVideoRecorder.setMusic(String path,long startTime,long duration);
//移除背景音乐
AliyunIVideoRecorder.removeMusic();
```

水印

```
//添加水印
AliyunIVideoRecorder.addWaterMark(EffectImage effectImage);
//删除水印
AliyunIVideoRecorder.removeWaterMark(EffectImage effectImage);
```

设置背景

```
//添加背景颜色
AliyunIVideoRecorder.setBackgroundColor(int color);
//添加背景图片bitmap
AliyunIVideoRecorder.setBackgroundImage(Bitmap bitmap);
//添加背景图片路径
AliyunIVideoRecorder.setBackgroundImage(String path);
//清除背景
AliyunIVideoRecorder.clearBackground();
```

多源录制代码示例

```
import android.os.Bundle;
import android.os.Environment;
import android.view.SurfaceView;
import android.view.View;
import android.widget.TextView;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.aliyun.svideosdk.common.struct.common.AliyunLayoutParam;
import com.aliyun.svideosdk.common.struct.common.VideoDisplayMode;
import com.aliyun.svideosdk.common.struct.encoder.VideoCodecs;
import com.aliyun.svideosdk.common.struct.recorder.CameraType;
import com.aliyun.svideosdk.multirecorder.AliyunICameraCapture;
import com.aliyun.svideosdk.multirecorder.AliyunIVideoRecorder;
import com.aliyun.svideosdk.multirecorder.AliyunIViewCapture;
import com.aliyun.svideosdk.multirecorder.OnVideoRecordListener;
import com.aliyun.svideosdk.multirecorder.config.AliyunVideoRecorderConfig;
import com.aliyun.svideosdk.multirecorder.impl.AliyunMultiRecorderCreator;
import java.io.File;
/**
 * 多源录制示例demo
 * 注意：需提前开启存储、相机、麦克风3个手机权限
 */
public class MultiRecorderDemo extends AppCompatActivity {
    private AliyunIVideoRecorder mRecorder;
    private TextView mBtnRecord;
    private SurfaceView mCameraPreview;
    private View mViewRecord;
    private View mViewRecordIcon;
    private boolean mIsRecording = false;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.multi_recoder_demo_layout);
        initViews();
        initRecorder();
```

```
        initRecorder();
    }
    @Override
    protected void onResume() {
        super.onResume();
        mRecorder.startPreview();
    }
    @Override
    protected void onPause() {
        super.onPause();
        stopRecording();
        mRecorder.stopPreview();
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mRecorder.destroy();
    }
    private String getSaveDir() {
        String saveDir = Environment.getExternalStorageDirectory().getAbsolutePath() + File
.separator + "MultiRecord";
        File dirFile = new File(saveDir);
        if (!dirFile.exists()) {
            dirFile.mkdirs();
        }
        return saveDir;
    }
    private void initViews() {
        mBtnRecord = findViewById(R.id.record_btn);
        mCameraPreview = findViewById(R.id.multi_record_camera);
        mViewRecord = findViewById(R.id.multi_record_view);
        mViewRecordIcon = mViewRecord.findViewById(R.id.record_view_icon);
    }
    private void initRecorder() {
        //1. 参数配置
        AliyunVideoRecorderConfig config = AliyunVideoRecorderConfig.builder()
                //必需设置的参数
                .videoWidth(1080)
                .videoHeight(1920)
                .outputPath(getSaveDir() + File.separator + System.currentTimeMillis() + ".
mp4")
                //其他非必须设置的参数
                .videoCodecs(VideoCodecs.H264_HARDWARE) //硬编
                .build();
        //2.创建录制接口
        mRecorder = AliyunMultiRecorderCreator.getVideoRecorderInstance(this, config);
        //3. 采集源配置
        //3.1 摄像头录制
        addCameraCapture();
        //3.2 View录制
        addViewCapture();
        //3.3 采集源准备
        mRecorder.prepare();
        //4 回调设置
        mRecorder.setOnRecordListener(new OnVideoRecordListener() {
```

```
            @Override
            public void onProgress(long duration) {
            }
            @Override
            public void onFinish(String outputPath) {
            }
            @Override
            public void onClipComplete(boolean validClip, long clipDuration) {
            }
            @Override
            public void onMaxDuration() {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        stopRecording();
                    }
                });
            }
            @Override
            public void onError(int errorCode) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        stopRecording();
                    }
                });
            }
            @Override
            public void onInitReady() {
            }
        });
    }
    private void addCameraCapture() {
        //1.布局配置：左半边
        AliyunLayoutParam cameraLayoutParam = AliyunLayoutParam.builder()
                .layoutLevel(1)
                .centerX(0.25f)
                .centerY(0.5f)
                .widthRatio(0.5f)
                .heightRatio(1.0f)
                .displayMode(VideoDisplayMode.FILL)
                .build();
        //2.添加采集源
        AliyunICameraCapture cameraCapture = mRecorder.getVideoCapture().addCameraCapture(c
ameraLayoutParam);
        //3.配置采集源
        //设置摄像头预览view，必需属性
        cameraCapture.setDisplayView(mCameraPreview);
        //其他设置（按需设置）
        cameraCapture.setCamera(CameraType.BACK);//后置摄像头
    }
    private void addViewCapture() {
        //1.布局配置：右半边
        AliyunLayoutParam viewLayoutParam = AliyunLayoutParam.builder()
```

```
                .layoutLevel(2)
                .centerX(0.75f)
                .centerY(0.5f)
                .widthRatio(0.5f)
                .heightRatio(1.0f)
                .displayMode(VideoDisplayMode.FILL)
                .build();
        //2.添加采集源
        // 获取目前录制View
        View recordView = mViewRecord;
        AliyunIViewCapture viewCapture = mRecorder.getVideoCapture().addViewCapture(viewLay
outParam, recordView);
        //3.配置采集源（按需设置）
    }
    private void startRecording() {
        if (mIsRecording) {
            return;
        }
        mIsRecording = true;
        mRecorder.startRecording();
    }
    private void stopRecording() {
        if (!mIsRecording) {
            return;
        }
        mRecorder.stopRecording();
        mIsRecording = false;
    }
    public void onClickRecord(View view) {
        if (mIsRecording) {
            stopRecording();
            mBtnRecord.setText("开始录制");
        } else {
            startRecording();
            mBtnRecord.setText("停止录制");
        }
    }
}
```

XML文件配置示例

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/demo_root_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
        <SurfaceView
            android:id="@+id/multi_record_camera"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1" />
        <FrameLayout
            android:id="@+id/multi_record_view"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1">
            <TextView
                android:id="@+id/record_view_icon"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:text="View录制" />
        </FrameLayout>
    </LinearLayout>
    <Button
        android:id="@+id/record_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginLeft="5dp"
        android:layout_marginBottom="5dp"
        android:onClick="onClickRecord"
        android:text="开始录制" />
</RelativeLayout>
```

4.5. 视频裁剪

短视频SDK提供了裁剪模块，支持对视频按时长、画幅裁剪，对音频按时长裁剪，对图片按画幅裁剪。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

名称	功能
AliyunICrop	裁剪功能核心类，包括裁剪、设置裁剪参数、设置回调等裁剪核心功能。
AliyunCropCreator	裁剪工厂类，获取裁剪实例。
CropParam	裁剪参数，设置裁剪的宽高、裁剪模式、输出路径等参数。
CropCallback	裁剪回调，设置裁剪完成、裁剪进度、裁剪失败等回调。

裁剪流程

流程	说明	示例代码
1	创建裁剪实例。	创建实例
2	设置裁剪的宽高、裁剪模式、输出路径等参数。	设置裁剪参数
3	设置裁剪完成、裁剪进度、裁剪失败等回调。	设置回调
4	开始裁剪。	开始裁剪
5	释放资源。	释放资源

创建实例

创建裁剪实例。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

创建实例

```
AliyunICrop cropper = AliyunCropCreator.createCropInstance(context);
```

设置裁剪参数

设置裁剪的宽高、裁剪模式、输出路径等参数。设置裁剪参数时，必选参数为setOutputWidth()、setOutputHeight()、setOutputPath()、setInputPath()。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//设置裁剪参数
CropParam cropParam = new CropParam();
//裁剪后宽度，单位：像素
cropParam.setOutputWidth(720);
//裁剪后高度，单位：像素
cropParam.setOutputHeight(1080);
 //设置裁剪模式，Scale：等比缩放裁剪， Fill：填充模式
cropParam.setScaleMode(VideoDisplayMode.Scale);
//输出文件路径
cropParam.setOutputPath(outputPath);
//视频源文件路径
cropParam.setInputPath(inputPath);
//开始时间，单位：微秒
cropParam.setStartTime(startTime);
//结束时间，单位：微秒
cropParam.setEndTime(endTime);
//裁剪媒体类型，包括图片、视频、音频
cropParam.setMediaType(MediaType.ANY_VIDEO_TYPE);
//裁剪矩阵
int startCropPosX = 0;
int startCropPoxY = 0;
Rect cropRect = new Rect(startCropPosX, startCropPoxY, startCropPosX + outputWidth,
    startCropPoxY + outputHeight);
cropParam.setCropRect(cropRect);
//帧率
cropParam.setFrameRate(30);
//gop
cropParam.setGop(5);
//视频质量
cropParam.setQuality(VideoQuality.HD);
//视频编码方式
cropParam.setVideoCodec(VideoCodecs.H264_HARDWARE);
//填充颜色
cropParam.setFillColor(Color.BLACK);
cropper.setCropParam(cropParam);
```

设置回调

设置裁剪完成、裁剪进度、裁剪失败等回调。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
设置回调
cropper.setCropCallback(new CropCallback() {
    @Override
    public void onProgress(int i) {
      //裁剪进度
    }
    @Override
    public void onError(int i) {
      //错误代码
        Log.i("TestCrop", "testCrop,onError:" + i);
    }
    @Override
    public void onComplete(long executeTime) {
      //裁剪完成
    }
    @Override
    public void onCancelComplete() {
    }
});
```

开始裁剪

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
开始裁剪
cropper.startCrop();
```

释放资源

裁剪完成后，销毁接口，释放资源。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
释放资源
cropper.dispose();
```

视频裁剪示例代码

```
//1.实例创建
AliyunICrop aliyunICrop = AliyunCropCreator.createCropInstance(context);
CropParam cropParam = new CropParam();
//2.设置裁剪参数
//必要参数-输出视频宽高，源文件路径，输出文件路径
int outputWidth = 720;
int outputHeight = 1080;
cropParam.setOutputWidth(outputWidth);
cropParam.setOutputHeight(outputHeight);
cropParam.setOutputPath("/storage/emulated/0/DCIM/Camera/test.mp4");
cropParam.setInputPath("/storage/emulated/0/DCIM/Camera/lesson-01.mp4");
//可选参数如下：
//媒体类型，默认为 ANY_VIDEO_TYPE
cropParam.setMediaType(MediaType.ANY_VIDEO_TYPE);
//裁剪矩阵
int startCropPosX = 0;
int startCropPoxY = 0;
Rect cropRect = new Rect(startCropPosX, startCropPoxY, startCropPosX + outputWidth,
    startCropPoxY + outputHeight);
cropParam.setCropRect(cropRect);
//VideoDisplayMode.SCALE 缩放 VideoDisplayMode.FILL 填充
cropParam.setScaleMode(VideoDisplayMode.SCALE);
//帧率
cropParam.setFrameRate(30);
//gop
cropParam.setGop(5);
//视频质量
cropParam.setQuality(VideoQuality.HD);
//视频编码方式
cropParam.setVideoCodec(VideoCodecs.H264_HARDWARE);
//填充颜色
cropParam.setFillColor(Color.BLACK);
aliyunICrop.setCropParam(cropParam);
//3.设置裁剪回调
aliyunICrop.setCropCallback(new CropCallback() {
    @Override
    public void onProgress(int i) {
    }
    @Override
    public void onError(int i) {
        Log.i("TestCrop", "testCrop,onError:" + i);
    }
    @Override
    public void onComplete(long l) {
        Log.i("TestCrop", "onComplete:" + l);
        Toast.makeText(context, "裁剪完成，耗时为:" + l, Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onCancelComplete() {
    }
});
//4.开始裁剪
aliyunICrop.startCrop();
```

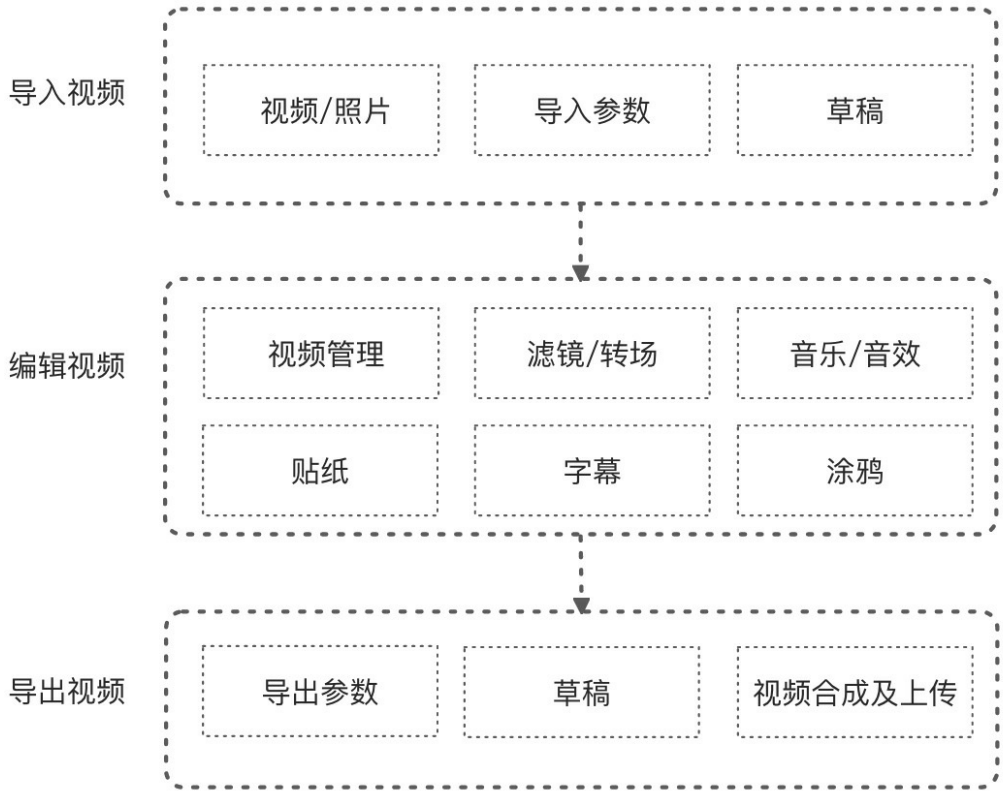
4.6. 视频编辑

4.6.1. 概述

短视频SDK提供视频编辑与导出功能，支持视频图片素材混合导入，提供滤镜、配音、时间特效、过渡效果等丰富的编辑效果。本文介绍视频编辑的流程及版本差异。

总体流程

编辑制作一个视频通常需要先导入视频，再编辑视频，最后导出视频。总体的流程及各环节涉及的功能点如下图所示：



版本支持

各版本支持的功能如下表所示，表格中的✓表示支持，×表示不支持。

功能	基础版	标准版	专业版
导入视频	×	✓	✓
导出视频	×	✓	✓
视频管理（编辑视频）	×	✓	✓

功能	基础版	标准版	专业版
音乐及音效（编辑视频）	×	✓	✓
滤镜及转场（编辑视频）	×	✓	✓
滤镜及转场（编辑视频）	×	✓	✓
时间特效（编辑视频）	×	✓	✓
画中画（编辑视频）	×	✓	✓
字幕（编辑视频）	×	×	✓
动态贴纸（编辑视频）	×	×	✓
MV（编辑视频）	×	×	✓

4.6.2. 导入视频

短视频SDK支持视频图片素材混合导入，支持本地视频导入和草稿箱导入这两种方式。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

相关类功能

类名	功能
AliyunIImport	视频导入核心类，包括生成配置、释放资源等。
AliyunImportCreator	工厂类，获取导入实例。
AliyunImageClip	图片片段，用于获取图片的宽高等信息。
AliyunVideoClip	视频片段，用于获取视频的开始结束时间、视频宽高等信息。
AliyunVideoParam	视频输出参数，设置输出视频的宽高、显示模式、缩放比等参数。

本地视频导入

本地视频导入，即通过AliyunIImport类添加不同的视频片段，最终生成视频配置的URI，作为AliyunIEditor类的输入参数。

```
//1.创建实例
AliyunIImport importor = AliyunImportCreator.getImportInstance(context);
//2.1添加视频
importor.addMediaClip(new AliyunVideoClip.Builder()
    .source(filePath)
    .startTime(startTime)
    .endTime(startTime + duration)
    .duration(duration)
    .build());
//2.2 添加图片
importor.addMediaClip(new AliyunImageClip.Builder()
    .source(filePath)
    .duration(duration)
    .build());
//3. 设置输出参数
AliyunVideoParam param = new AliyunVideoParam.Builder()
        .frameRate(frameRate) //帧率
        .gop(gop) // gop
        .crf(crf) // crf
        .videoQuality(videoQuality) //视频质量
        .scaleMode(mScaleMode) //缩放模式
        .outputWidth(outputWidth) //视频宽度
        .outputHeight(outputHeight) //视频高度
        .videoCodec(videoCodec)
        .build();
importor.setVideoParam(param);
//4.生成视频源配置文件
//生成配置文件Uri，作为AliyunIEditor的输入参数
String videoConfigPath = import.generateProjectConfigure();
//5.用完后销毁
importor.release()
```

草稿箱导入

草稿箱导入，即将草稿箱的URI作为AliyunIEditor类的输入参数来实现导入视频，详细操作请参见草稿箱。

4.6.3. 编辑视频

短视频SDK提供视频编辑与导出功能，支持视频图片素材混合导入、提供滤镜、配音、时间特效等丰富的编辑效果。

版本支持

版本	是否支持
专业版	支持所有功能。
标准版	部分支持，支持除了字幕、动态贴纸、MV以外的其他功能。
基础版	不支持。

相关类功能

操作	类名	功能
初始化	AliyunIEditor	视频编辑核心类。
	AliyunEditorFactory	工厂类。
视频管理	AliyunIClipConstructor	视频源管理器。
	AliyunImageClip	图片片段。
	AliyunVideoClip	视频片段。
设置视频特效	EffectBean	滤镜及MV的Model。
	EffectFilter	动效滤镜Model。
	TransitionBase	转场Model。
设置画中画	AliyunIPipManager	画中画管理类，负责画中画的增删改查。
	AliyunIPipController	画中画控制器，设置画中画的开始时间，结束时间。
	AliyunIPipTrack	轨道信息获取。
	AliyunILayoutController	布局控制器，对画中画移动、缩放、旋转、透明度等设置。
	AliyunIAnimationController	动画控制器，可以对画中画做帧动画。
	AliyunIAudioController	声音控制器，控制画中画音量，降噪、音效、淡入淡出等。
	AliyunIAugmentationController	画面调节控制器，可以调节饱和度、亮度、对比度等。
设置字幕及动态贴纸	AliyunPasterManager	字幕及动态贴纸管理器。
	AliyunPasterControllerCompoundCaption	字幕控制器。
	AliyunPasterController	动态贴纸控制器。
草稿箱	AliyunEditorProject	工程配置。
	AliyunDraftManager	草稿管理。
	AliyunDraft	草稿。
	AliyunDraftResourceLoader	资源加载器。
	AliyunDraftResourceUploader	资源上传器。

操作	类名	功能
	AliyunDraftResourceDownloader	资源下载器。
	AliyunDraftResTask	草稿资源处理任务。

编辑视频流程

阶段	流程	说明	示例代码
基础	1	创建并初始化编辑器。	初始化
	2	在编辑过程中动态裁剪视频、动态更换视频源、动态调整视频转场时间及转场效果。编辑核心类是AliyunIEditor类。	视频管理
	3	设置编辑器的预览播放。	预览控制
进阶	4	设置滤镜、转场及MV特效。	设置视频特效
	5	设置背景音乐、配音及音效。	设置音乐及音效
	6	设置画中画。	设置画中画
	7	设置字幕、花字、文字气泡及动态贴纸。	设置字幕及动态贴纸
	8	支持编辑草稿箱中的视频、或将编辑完成的视频保存至草稿箱。	草稿箱
	9	设置时间特效、水印及涂鸦。	其他设置

初始化

创建并初始化编辑器。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//1. 实例化
//configPath为导入视频的地址，为草稿箱地址或者AliyunIImport.generateProjectConfigure()重新生成后的地址
Uri uri = Uri.parse(configPath);
AliyunIEditor editor = AliyunEditorFactory.creatAliyunEditor(uri, null);
//2. 初始化
//初始化并设置预览窗口
editor.init(surfaceView, context);
//3. 销毁，当不再使用时需要调用销毁接口
editor.onDestroy()
```

视频管理

视频编辑器里的视频或图片最终由视频源管理器AliyunIClipConstructor统一管理，通过AliyunIClipConstructor修改编辑器里的视频或图片后，需要手动调用AliyunIEditor.applySourceChange()应用更新的视频或图片。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

视频源管理操作

```
//1.获取视频源管理器
AliyunIClipConstructor contructor = AliyunIEditor.getSourcePartManager();
//[可选] 2.添加视频或图片
//在时间轴最后添加视频或图片，其中视频用AliyunVideoClip，图片用AliyunImageClipAliyunImageClip
contructor.addMediaClip(AliyunClip clip);
//在指定位置的添加视频或图片，其中视频用AliyunVideoClip，图片用AliyunImageClipAliyunImageClip
contructor.addMediaClip(int index, AliyunClip clip);
//[可选] 3.删除视频或图片
//删除最后一个视频或图片
contructor.deleteMediaClip();
//删除某一个指定的视频或图片
contructor.deleteMediaClip(int index);
//[可选] 4.替换视频或图片
//替换指定位置视频或图片
contructor.updateMediaClip(int index, AliyunClip clip);
//替换所有的视频或图片
contructor.updateAllClips(List<AliyunClip> clips);
//5.应用更新源：视频源操作完成后，要调用该方法应用更新，操作才会生效
AliyunIEditor.applySourceChange();
```

其他相关操作

```
//交换视频源顺序
contructor.swap(int pos1, int pos2);
//获取当前视频源个数
contructor.getMediaPartCount();
//获取当前视频源列表
contructor.getAllClips();
```

预览控制

在视频编辑过程中，提供一系列对当前视频的播放控制操作，如播放、暂停、获取当前时长等。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//开始播放
AliyunIEditor.play();
//继续播放
AliyunIEditor.resume();
// 暂停播放
AliyunIEditor.pause();
// 跳转到指定位置
AliyunIEditor.seek(long time);
// 设置静音播放
AliyunIEditor.setAudioSilence(boolean silence);
// 设置音量
AliyunIEditor.setVolume(int volume);
// 设置视频显示模式
AliyunIEditor.setDisplayMode(VideoDisplayMode mode);
// 填充模式下设置填充背景色
AliyunIEditor.setFillBackgroundColor(int color);
//获取当前流的位置 - 不受时间特效影响
AliyunIEditor.getCurrentStreamPosition();
// 获取当前播放的位置 - 受时间特效影响
AliyunIEditor.getCurrentPlayPosition();
// 获取流时长 - 不受时间特效影响
AliyunIEditor.getStreamDuration();
// 获取播放总时长 - 受时间特效影响
AliyunIEditor.getDuration();
```

设置视频特效

目前支持设置的视频特效包括滤镜、转场和MV，支持自定义制作，制作方法请参见滤镜及转场和MV。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

- 滤镜分为lut滤镜、静态滤镜和动效滤镜。
  - lut滤镜：使用Lookup Table方式进行像素替换。
  - 静态滤镜：通过编写着色语言方式进行像素计算，不支持带动画效果。Bean为EffectBean，接口参数请参考EffectBean。
  - 动效滤镜：通过编写着色语言方式进行像素计算，带动画效果。Bean为EffectFilter，接口参数请参考EffectFilter。
- 目前短视频SDK提供的转场效果包括：TransitionCircle（圆形打开）、TransitionFade（淡入淡出）、TransitionFiveStar（五角星）、TransitionShutter（百叶窗）、TransitionTranslate（平移）。

lut滤镜

```
LUTEffectBean bean = new LUTEffectBean();
bean.setPath("image_01.png");
bean.setIntensity(1.f);
//添加lut滤镜
mAliyunIEditor.applyLutFilter(bean);
//删除lut滤镜
mAliyunIEditor.applyLutFilter(null);
```

静态滤镜

```
EffectBean effect = new EffectBean();
effect.setId(id)
effect.setSource(new Souce(filePath));
//添加滤镜
AliyunIEditor.applyFilter(effect);
//删除滤镜
AliyunIEditor.applyFilter(new EffectBean());
```

动效滤镜

```
EffectFilter effectFilter = new EffectFilter(new Souce(filePath));
effectFilter.setStartTime(startTime);
effectFilter.setDuration(duration);
//添加动效滤镜
AliyunIEditor.addAnimationFilter(effectFilter);
//删除指定动效滤镜
AliyunIEditor.removeAnimationFilter(effectFilter);
//删除所有动效滤镜
AliyunIEditor.clearAllAnimationFilter();
```

转场

```
//1.设置转场
//设置一个转场，index为转场位置，从0开始记，取消转场的话，transition传null即可
AliyunEditor.setTransition(int index, TransitionBase transition);
//设置批量转场，取消转场的话，transition传null即可
AliyunEditor.setTransition(Map<Integer, TransitionBase> transitions); //设置多个转场
//2.更新转场
//从某个转场更新成另一个转场，transition不允许为null
AliyunEditor.updateTransition(int index, TransitionBase transition);
```

MV

支持自定义MV，MV的制作规范请参见MV。

```
//添加MV
AliyunIEditor.applyMV(EffectBean effect);
//删除MV
AliyunIEditor.applyMV(null);
```

设置音乐及音效

音乐

音乐分为背景音乐和配音。背景音乐不受时间特效影响（变速、重复、倒放等），而配音会受到时间特效的影响。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
EffectBean musicBean = new EffectBean();
musicBean.setId(effectInfo.id);
musicBean.setSource(effectInfo.getSource());
//切换音乐seek到0清音乐缓存，避免响一声
musicBean.setStartTime(startTime);
musicBean.setDuration(Integer.MAX_VALUE);//设置为最大时长
musicBean.setStreamStartTime(streamStartTime);
musicBean.setStreamDuration(streamDuration);
int audioSteamId;
//1.添加背景音乐/配音
audioSteamId = AliyunIEditor.applyMusic(musicBean);//音乐
AliyunIEditor.applyDub(EffectBean effect);//配音
//2.删除背景音乐/配音
AliyunIEditor.removeMusic(EffectBean effect);//音乐
AliyunIEditor.removeDub(EffectBean effect);//配音
//3.调整背景音乐/配音与原音的比重
AliyunIEditor.applyMusicMixWeight(int audioSteamId, int weight);
//4.调整指定音频流音量
//背景音乐、配音、原音，详细请查看API文档
AliyunIEditor.applyMusicWeight(int audioSteamId, int weight);
//5.指定音频流降噪
AliyunIEditor.denoise(int audioSteamId, boolean needDenoise);
```

音效

短视频SDK支持对每路音频流设置音效，目前提供的音效包括如下：

- AudioEffectType.EFFECT_TYPE_LOLITA（萝莉）
- AudioEffectType.EFFECT_TYPE_REVERB（混响）
- AudioEffectType.EFFECT_TYPE_UNCLE（大叔）
- AudioEffectType.EFFECT_TYPE_ECHO（回声）
- AudioEffectType.EFFECT_TYPE_ROBOT（机器人）
- AudioEffectType.EFFECT_TYPE_BIG_DEVIL（大魔王）
- AudioEffectType.EFFECT_TYPE_MINIONS（小黄人）
- AudioEffectType.EFFECT_TYPE_DIALECT（方言）

```
//1. 设置音效
//AudioEffectType的更多内容请查看API文档
int audioEffect(int audioSteamId, AudioEffectType type, int weight);
//2. 删除音效
//音效支持叠加操作，想要切换音效需要先删除上次设置的音效。
int removeAudioEffect(int audioSteamId, AudioEffectType type);
```

设置画中画

画中画功能允许在现用主轨道的基础上，添加一个或者多个画中画。

- 主轨道：编辑页面默认轨道，有且仅有一个主轨道，一个主轨道可以有多个视频流。
- 画中画：允许添加多个画中画，画中画允许设置位置，缩放，旋转等。创建画中画默认创建一个画中画轨道。画中画可以在不同画中画轨道中移动。

> **说明** 短视频SDK对画中画的个数未做限制，但建议同一时刻画面中最好不要超过3个画中画，具体限制个数由业务方自己做决定。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//从编辑主接口获取画中画管理类
AliyunIPipManager pipManager = mAliyunIEditor.getPipManager();
//增加画中画
mAliyunIEditor.puase(); //先暂停
long current = mAliyunIEditor.getCurrentPlayPosition(); //获取当前的播放时间点
AliyunIPipManager pipManager = mAliyunIEditor.getPipManager();
AliyunIPipController pipController = pipManager.createNewPip("流文件地址");
pipController.setTimelineStartTime(current) //从轨道当前时间点开始
                .setClipStartTime(0) //画中画视频本身开始时间
                .apply();  //应用生效
//删除画中画
mAliyunIEditor.puase(); //先暂停
AliyunIPipManager pipManager = mAliyunIEditor.getPipManager();
pipManager.removePip(pipController);
//修改布局：AliyunILayoutController
mAliyunIEditor.puase(); //先暂停
AliyunILayoutController layoutController = pipController.getLayoutController();  // 获取布局控制器
layoutController.setRotation(3.14)  //设置旋转弧度 0~3.14
                .setScale(0.3)  //设置缩放
                .setPosition(0.5, 0.5)  //设置位置居中
                .apply();
//获取画中画画布中的位置
RectF rectf = pipController.getPipRectFInCurrentScreen(); //获得画中画所在的矩形区域
int width = mSurfaceView.getWidth() * rectf.width();  //画中画在画布中的实际宽度
int height = mSurfaceView.getWidth() * rectf.height();  //画中画在画布中的实际高度
//修改声音相关：AliyunIAudioController
mAliyunIEditor.puase(); //先暂停
AliyunIAudioController audioController = pipController.getAudioController(); //获取声音控制器
audioController.setVolume(100) //设置音量最大
                .setAudioEffect(AudioEffectType.EFFECT_TYPE_LOLITA)  //设置声音特效为萝莉音
                .apply();  //应用
//设置画中画帧动画：AliyunIAnimationController
mAliyunIEditor.puase(); //先暂停
AliyunIAnimationController animationController = pipController.getAnimationController();
if (mActionTranslate == null) { //添加位移动画
    mActionTranslate = new ActionTranslate();
    mActionTranslate.setFromPointX(-1);
    mActionTranslate.setFromPointY(-1);
    mActionTranslate.setToPointX(1);
    mActionTranslate.setToPointY(1);
    mActionTranslate.setStartTime(pipController.getTimeLineStartTimeInMillis() * 1000);
    mActionTranslate.setDuration(pipController.getClipDurationInMillis() * 1000);
    animationController.addFrameAnimation(mActionTranslate);
} else {  //删除位移动画
    animationController.removeFrameAnimation(mActionTranslate);
    mActionTranslate = null;
}
```

```
//画面调节：AliyunIAugmentationController
mAliyunIEditor.puase(); //先暂停
AliyunIAugmentationController augmentationController = pipController.getAugmentationControl
ler();   //获取画面调节控制器
augmentationController.setVignette(0~1) //设置暗角
                        .setSharpness(0~1)  //设置锐化程度
                        .setSaturation(0~1)  //设置保护度
                        .apply();  //应用
//获取轨道信息：AliyunIPipTrack
//方式一： 从管理类获取所有轨道
AliyunIPipManager pipManager = mAliyunIEditor.getPipManager();
List<AliyunIPipTrack> pipTrackList = pipManager.getPipTracks(); //获取所有画中画轨道
//方式二： 获取当前控制器所在轨道
AliyunIPipTrack pipTrack = pipController.getOwnerTrack();
List<AliyunIPipController> pipControllers = pipTrack.getPipClips(); //获取当前轨道下所有画中画
片段控制器
```

设置字幕及动态贴纸

字幕及动态贴纸统一通过AliyunPasterManager进行管理，通过AliyunPasterManager获取到对应的AliyunIPasterController进行相应的操作。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

基于字幕，短视频SDK还提供了花字及气泡文字的特效。花字、气泡文字及动态贴纸的制作请参见花字和动图。

字幕

添加/删除字幕

```
Source fontSouce = null;
long startTime = 0L;
long duration = 20000L;
//1. 添加字幕
AliyunPasterControllerCompoundCaption captionController = pasterManager.addCaptionWithStart
Time('输入文字', null, fontSouce, startTime,duration);
//2. 删除字幕
controller.remove()
```

更新字幕属性

AliyunPasterControllerCompoundCaption包含了字幕的所有操作，详细请查看接口说明。每次更新完字幕属性后，必须调用AliyunPasterControllerCompoundCaption.apply()。

```
//设置颜色
captionController.setColor(AliyunColor color);
//设置字体
captionController.setFontPath(ISouce fontPath);
//应用以上更新
captionController.apply();
```

花字

```
Source fontEffectSource(fontEffectFolder);
//1. 应用花字
captionController.setFontEffectTemplate(fontEffectSource)
//2. 取消花字
captionController.setFontEffectTemplate(null)
```

文字气泡

```
Source bubbleEffectSource(bubbleEffectFolder);
//1. 应用气泡
captionController.setBubbleEffectTemplate(bubbleEffectSource)
//2. 取消气泡
captionController.setBubbleEffectTemplate(null)
```

动态贴纸

添加动态贴纸

```
AliyunPasterController pasterController = pasterManager.addPasterWithStartTime(Source path,
long startTime, long duration);
```

设置动态贴纸属性

动态贴纸使用与字幕有所差异，动态贴纸是Android去展示的动画效果，因此设置贴纸的属性需要在Android实现UI（AliyunPasterBaseView），定义动态贴纸的大小、宽高、旋转角度等属性。由于是在平台层也同时实现了AliyunPasterBaseView，因此，提供了从渲染层展示、隐藏动态贴纸的操作，避免两者重叠。

```
//必须调用
pasterController.setPasterView(AliyunPasterBaseView pasterView);
//显示贴纸
pasterController.editCompleted();
//隐藏贴纸
pasterController.editStart();
```

贴纸效果图预览

```
//设置预览
pasterController.createPasterPlayer(TextureView view);
//同时手动触发播放/停止预览效果
protected void playPasterEffect() {
        TextureView pv = new TextureView(mPasterView.getContext());
        animPlayerView = mController.createPasterPlayer(pv);
        ViewGroup.LayoutParams lp = new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH
_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT);
        ViewGroup vg = (ViewGroup) mPasterView.getContentView();
        vg.addView(pv, 0, lp);
    }
    protected void stopPasterEffect() {
        ViewGroup vg = (ViewGroup) mPasterView.getContentView();
        vg.removeViewAt(0);
        animPlayerView = null;
    }
```

删除动态贴纸

```
pasterController.removePaster();
```

草稿箱

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

工程配置信息获取

```
//初始化
AliyunIEditor.init(SurfaceView surfaceView, Context context);
//获取配置信息
AliyunEditorProject project = AliyunIEditor.getEditorProject();
```

初始化草稿管理器

```
AliyunDraftManager draftManager = AliyunDraftManager.getInstance(context);
```

获取草稿列表

```
//异步获取草稿列表
AliyunDraftManager.getInstance(getContext())
                        .getDraftListByAsync(new AliyunDraftListCallback() {
                            @Override
                            public void onFailure(final String msg) {
                                //获取列表异常
                            }
                            @Override
                            public void onSuccess(final List<AliyunDraft> draftList) {
                                //草稿列表回调
                            }
                        });
```

根据草稿ID删除草稿

```
//删除草稿，draft为草稿列表单个item(草稿列表通过草稿列表接口获取)
AliyunDraftManager.getInstance(v.getContext()).deleteDraft(draft.getId());
```

草稿重命名

```
//重命名，draft为草稿列表单个item(草稿列表通过草稿列表接口获取)
AliyunDraftManager.getInstance(v.getContext()).rename(draft.getId(), newName);
```

草稿复制

```
//复制后会返回新草稿，draft为草稿列表单个item(草稿列表通过草稿列表接口获取)
AliyunDraft newDraft = AliyunDraftManager.getInstance(v.getContext()).copy(draft.getId());
```

草稿加载

```
//draft为草稿列表单个item(草稿列表通过草稿列表接口获取)
AliyunDraftManager.getInstance(v.getContext()).preLoadDraft(draft, new AliyunDraftResourceL
oader() {
                @Override
                public void onHandleResourceTasks(final List<AliyunDraftResTask> tasks) {
                    //缺少相关资源，返回需要处理的资源任务，必须对任务进行处理，可选项：修复、忽略、
删除
                    HashMap<String, List<AliyunDraftResTask>> map = new HashMap<>();
                    for (AliyunDraftResTask task : tasks) {
                        if (task.getSource() != null && !StringUtils.isEmpty(task.getSource
().getURL())) {
                            if (map.containsKey(task.getSource().getURL())) {
                                map.get(task.getSource().getURL()).add(task);
                            } else {
                                List<AliyunDraftResTask> list = new ArrayList<>();
                                list.add(task);
                                map.put(task.getSource().getURL(), list);
                            }
                        } else {
                            //必须对任务进行处理，可选项：修复、忽略、删除
                            if (task.getResModuleType() == AliyunResModuleType.MAIN_VIDEO)
{
                                task.getSource().setPath(EditorCommon.SD_DIR + "svideo_res/
image/aliyun_svideo_failed.jpg");
                                task.onHandleCallback(task.getSource());
                            } else if(task.getResModuleType() == AliyunResModuleType.TRANSI
TION) {
                                //删除
                                task.onRemove();
                            } else {
                                //忽略
                                task.onIgnore();
                            }
                        }
                        for (final Map.Entry<String, List<AliyunDraftResTask>> entry : map.
entrySet()) {
```

```
                            //key为资源地址，Value为对应资源地址需要处理的任务
                            final List<AliyunDraftResTask> list = entry.getValue();
                            try {
                                final String url = entry.getKey();
                                //判断是否是平台资源
                                if (url.startsWith(AlivcResUtil.SCHEME)) {
                                    //平台资源加载回调封装
                                    AlivcResUtil.LoadCallback callback = new AlivcResUtil.L
oadCallback() {
                                        @Override
                                        public void onSuccess(String path) {
                                            for (AliyunDraftResTask task : list) {
                                                Source source = task.getSource();
                                                source.setPath(path);
                                                task.onHandleCallback(source);
                                            }
                                        }
                                        @Override
                                        public void onFailure(String type, String msg) {
                                            Log.d("CloudDraft", "loadRes>Failure>type>" + t
ype + ">msg>" + msg);

                                            for (AliyunDraftResTask task : list) {
                                                task.onIgnore();
                                            }
                                        }
                                    };
                                    //加载平台资源，具体代码看demo
                                    AlivcResUtil.loadRes(context, url, callback);
                                } else {
                                    //下载用户资源，具体代码看demo
                                    downloadRes(url, new File(item.getEditorProjectUri()).g
etParent(), list);
                                }
                            } catch (Exception e) {
                                //出错
                                for (AliyunDraftResTask item : list) {
                                    item.onIgnore();
                                }
                            }
                        }
                    }
                }
                @Override
                public void onFailure(final String msg) {
                    //预加载失败
                    Toast.makeText(v.getContext(), "预加载失败", Toast.LENGTH_SHORT).show();
                }
                @Override
                public void onSuccess() {
                    //预加载处理成功即可进入编辑界面，draft为草稿列表单个item(草稿列表通过草稿列表接
口获取)，通过draft.getEditorProjectUri()来加载草稿
                    EditorActivity.startEdit(v.getContext(), draft);
                }
            });
```

上传草稿

上传草稿需配合草稿服务端，服务端简单代码示例下载。

```
//draft为草稿列表单个item(草稿列表通过草稿列表接口获取)
AliyunDraftManager.getInstance(context)
                          .uploadDraft(draft, new AliyunDraftResourceUploader() {
                              @Override
                              public void onHandleResourceTasks(final List<AliyunDraftResTa
sk> tasks) {
                                  //需要处理的上传资源任务
                                  HashMap<String, List<AliyunDraftResTask>> map = new HashM
ap<>();
                                  //过滤重复资源
                                  for (AliyunDraftResTask task : tasks) {
                                      if (task.getSource() == null) {
                                          task.onIgnore();
                                          continue;
                                      }
                                      //URL为空或者不以alivc_resource开头需要做上传处理
                                      String url = task.getSource().getURL();
                                      if (StringUtils.isEmpty(url) || !url.startsWith("aliv
c_resource")) {
                                          if (map.containsKey(task.getSource().getPath()))
{
                                              map.get(task.getSource().getPath()).add(task)
;
                                          } else {
                                              List<AliyunDraftResTask> list = new ArrayList
<>();
                                              list.add(task);
                                              map.put(task.getSource().getPath(), list);
                                          }
                                      } else {
                                          //忽略出错
                                          task.onIgnore();
                                      }
                                  }
                                  for (Map.Entry<String, List<AliyunDraftResTask>> entry :
map.entrySet()) {
                                      try {
                                          for (AliyunDraftResTask task : tasks) {
                                              Source source = task.getSource();
                                              //上传成功后回调远程地址
                                              source.setURL();
                                              task.onHandleCallback(source);
                                          }
                                      } catch (Exception e) {
                                          //忽略出错
                                          List<AliyunDraftResTask> list = entry.getValue();
                                          for (AliyunDraftResTask item:list){
                                              item.onIgnore();
                                          }
```

```
                                    }
                                }
                            }
                            @Override
                            public void onSuccess(final String projectPath, String coverU
rl) {
                                //资源全部上传成功后，返回工程配置地址和封面地址
                                //可自行处理上传到云端，其他用户就可以通过工程配置地址恢复草稿到
编辑状态
                            }
                            @Override
                            public void onFailure(final String msg) {
                                Toast.makeText(context,"备份失败",Toast.LENGTH_SHORT).show
();
                            }
                        });
```

下载草稿

下载草稿需配合草稿服务端，服务端简单代码示例下载。

```
//根据草稿工程配置下载草稿相关资源，file为工程配置文件(草稿备份后从服务器下载下来的)
AliyunDraftManager.getInstance(context).downloadDraft(file, new AliyunDraftResourceDownload
er() {
            @Override
            public void onHandleResourceTasks(final String projectDir, final List<AliyunDra
ftResTask> tasks) {
                //待处理的草稿资源任务，需要下载草稿里面包含的资源
                HashMap<String, List<AliyunDraftResTask>> map = new HashMap<>();
                //过滤重复资源
                for (AliyunDraftResTask task : tasks) {
                    if (task.getSource() == null || StringUtils.isEmpty(task.getSource().ge
tURL())) {
                        task.onIgnore();
                    } else if (map.containsKey(task.getSource().getURL())) {
                        map.get(task.getSource().getURL()).add(task);
                    } else {
                        List<AliyunDraftResTask> list = new ArrayList<>();
                        list.add(task);
                        map.put(task.getSource().getURL(), list);
                    }
                }
                for (final Map.Entry<String, List<AliyunDraftResTask>> entry : map.entrySet
()) {
                    final List<AliyunDraftResTask> list = entry.getValue();
                    try {
                        final String url = entry.getKey();
                        //这里需要根据url下载草稿资源
                        for (AliyunDraftResTask task : list) {
                            Source source = task.getSource();
                            //下载完成后回调资源本地地址
                            source.setPath(path);
                            //如果是MV则解压出ID赋值给Source供显示还原
                            if (task.getResModuleType() == AliyunResModuleType.MV) {
                                try {
```

```
                                try {
                                    source.setId(Uri.parse(url).getQueryParameter("gid"));
                                }catch (Exception ignored){
                                }
                            }
                            task.onHandleCallback(source);
                        }
                    } catch (Exception e) {
                        //出错
                        for (AliyunDraftResTask item : list) {
                            item.onIgnore();
                        }
                    }
                }
            }
            @Override
            public void onSuccess(final AliyunDraft draft) {
                //资源全部下载完成后设置服务端ProjectID到本地草稿用来关联云端草稿
                AliyunDraftManager.getInstance(context).setProjectId(draft.getId(), project
Id);
                Toast.makeText(context,"成功恢复到本地",Toast.LENGTH_SHORT).show();
                //恢复成功后就可以在本地草稿列表查看
            }
            @Override
            public void onFailure(final String msg) {
                Toast.makeText(context,"恢复到本地失败",Toast.LENGTH_SHORT).show();
            }
        });
```

其他设置

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

时间特效

```
// 1.变速
//3.7.0版本开始变速接口可以针对多段视频/图片添加
int effectId;
effectId = AliyunIEditor.rate(float rate, long startTime, long duration, boolean needOrigin
Duration);
// 2.反复
effectId = AliyunIEditor.repeat(int times, long startTime, long duration, boolean needOrigi
nDuration);
//3.倒放
//特别注意：对于GOP大于5的视频需要先转码，否则将无法倒播。对于视频GOP的检查可以用NativeParser.getMax
GopSize()来获取。转码时将GOP SIZE设置为1，即CropParam.setGop(1);
effectId = AliyunIEditor.invert();
// 删除时间特效
AliyunIEditor.deleteTimeEffect(effectId);
```

水印

水印分为普通水印及片尾水印。普通水印会贯穿整个视频流时长，而片尾水印会在视频流结尾处添加水印。

```
//普通水印
AliyunIEditor.applyWaterMark(String imgPath, float sizeX, float sizeY, float posX, float po
sY);
//片尾水印
AliyunIEditor.addTailWaterMark(String imagePath, float sizeX, float sizeY, float posX, floa
t posY, long durationUs);
```

涂鸦

短视频SDK封装了一套涂鸦接口，包含画板、画笔等，整个涂鸦操作由涂鸦控制器（AliyunICanvasController）完成。

- 画板：涂鸦对应的UI交互View，可以添加到UI交互的ViewGroup。
- 画笔：一个android.graphics.Paint对象，开发者可以从外部设置，也可以使用默认画笔。

```
// 获取涂鸦控制器
int width = 600;
int height = 800
AliyunICanvasController controller = AliyunIEditor.obtainCanvasController( context, width,
height);
// 获取涂鸦画板
View canvasView = controller.getCanvas();
// 进行涂鸦
// 应用涂鸦
controller.applyPaintCanvas();
// 释放资源
controller.release();
//*******
//其他操作
//*******
// 撤销上一笔
controller.undo();
// 清除画布
controller.clear();
// 移除涂鸦
controller.removeCanvas();
// 判断是否由涂鸦
controller.hasCanvasPath();
```

4.6.4. 导出视频

导出视频包含视频配置导出、视频合成及上传。本文为您介绍导出视频的流程以及方法。

版本支持

版本	是否支持
专业版	支持
标准版	支持

版本	是否支持
基础版	支持

相关类功能

类名	功能
AliyunVideoParam	视频输出参数，设置合成导出视频的参数。
AliyunIVodCompose	视频合成上传类，包括初始化、合成视频、上传视频、更新上传凭证等视频合成及上传相关功能。
AliyunComposeFactory	工厂类，创建合成实例。

导出视频流程

流程	说明	示例代码
1	导出当前编辑视频的配置。	视频配置导出
2	合成视频，支持暂停、恢复及取消合成。	视频合成
3	上传视频，支持暂停、继续及取消上传。	视频上传
4	释放资源。	释放资源

说明

- 视频合成接口和上传接口的调用有顺序要求，合成接口调用完成后可以调用上传接口。
- 合成接口可多次调用，而上传接口只上传最后一次合成生成的视频文件。
- 如果编辑的对象是视频，在创建AliyunIVodCompose实例前，需要先将该视频的特效持久化到本地配置文件中，否则合成的视频无此特效。持久化编辑特效接口如下：

```
AliyunIEditor.saveEffectToLocal();
```

视频配置导出

导出当前编辑视频的配置，此时并不会合成输出最终的视频文件。

接口参数请参考AliyunVideoParam。

```
//视频配置导出，最终输出outputPath的配置文件
AliyunIEditor.compose(AliyunVideoParam param, String outputPath, AliyunIComposeCallBack callback);
//取消视频配置导出
AliyunIEditor.cancelCompose();
```

视频合成

阿里云短视频SDK提供的一套单独进行合成上传的功能接口，用来实现将编辑完的视频在另一界面合成上传，核心类AliyunIVodCompose封装了视频合成与上传功能，方便客户端更好地合成与上传视频。合成接口可多次调用。

接口参数请参考AliyunIVodCompose及AliyunComposeFactory。

初始化合成上传实例

```
//创建
AliyunComposeFactory.createAliyunVodCompose();
//初始化
AliyunIVodCompose.init(Context context);
```

合成

```
AliyunIVodCompose.compose(String config, String output, AliyunIComposeCallBack callback);
```

合成控制（按需使用）

```
//暂停合成
AliyunIVodCompose.pauseCompose();
//恢复合成
AliyunIVodCompose.resumeCompose();
//取消合成
AliyunIVodCompose.cancelCompose();
```

视频上传

视频合成完成后调用上传接口上传视频，上传接口只上传最后一次合成生成的视频文件。

接口参数请参考AliyunIVodCompose及AliyunComposeFactory。

上传

- 获取上传地址及上传凭证。

  AliyunIVodCompose是通过上传地址和上传凭证上传，在上传之前，需要获取上传地址及上传凭证，详细操作请参见获取音视频上传地址和凭证。

- 上传视频文件到OSS Bucket中。

  ```
  //通过上传地址和上传凭证上传视频
  //videoPath:视频文件地址
  //uploadAddress:上传地址
  //uploadAuth:上传凭证
  //aliyunVodUploadCallBack:上传的回调
  //上传视频
  AliyunIVodCompose.uploadVideoWithVod(String videoPath, String uploadAddress, String uploadAuth, AliyunIVodUploadCallBack aliyunVodUploadCallBack);
  ```

- 刷新视频上传凭证。

  考虑到上传凭证带有时效性，所以上传后，需要在过期回调onUploadTokenExpired方法中重新获取上传凭证上传，并通过AliyunIVodCompose.refreshWithUploadAuth(String uploadAuth);进行刷新，更多信息，请参见刷新视频上传凭证。

上传控制（按需使用）

```
//暂停上传
AliyunIVodCompose.pauseUpload();
//继续上传
AliyunIVodCompose.resumeUpload();
//取消上传
AliyunIVodCompose.cancelUpload();
```

释放资源

上传完成后，销毁接口，释放资源。

```
AliyunIVodCompose.release();
```

4.7. 视频模板（剪同款）

视频模板功能即通过替换模板中的视频、图片、文字，导出生成新的视频，是一种快速或规模化的视频生产方式。可通过短视频的模板能力实现剪同款功能。

版本支持

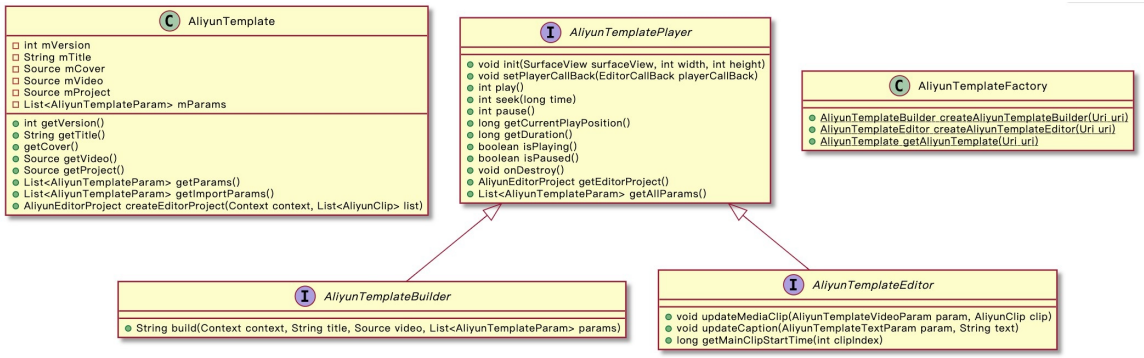
版本	是否支持
专业版	支持 **说明** 短视频SDK V3.27及以上版本才支持视频模板功能。
标准版	不支持
基础版	不支持

相关类功能

类名	功能
AliyunTemplate	模板。
AliyunTemplateParam	模板参数。
AliyunTemplateTextParam	模板文字参数。
AliyunTemplateImageParam	模板图片参数。
AliyunTemplateVideoParam	模板视频参数。
AliyunTemplatePlayer	模板播放器。
AliyunTemplateBuilder	模板构建器。

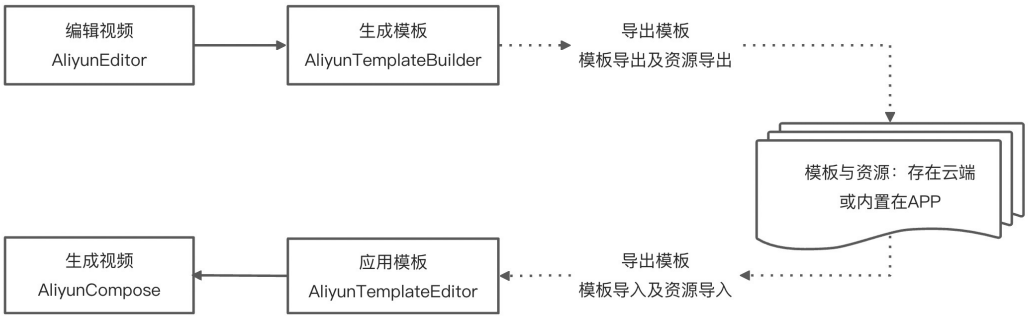
类名	功能
AliyunTemplateEditor	模板编辑器。
AliyunTemplateFactory	模板工厂。

相关类的关系如下图所示：



视频模板应用流程

视频模板的使用流程如下图所示：



生成模板

生成视频后可以选择可替换的节点（视频、图片、字幕）生成模板，调用 `AliyunTemplateBuilder` 接口生成模板。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//根据工程配置文件路径path生成模板构建器
AliyunTemplateBuilder mAliyunTemplateBuilder = AliyunTemplateFactory.createAliyunTemplateBu
ilder(Uri.parse(path));
//获取可编辑参数
List<AliyunTemplateParam> list = mAliyunTemplateBuilder.getAllParams();
File appFilesDir = getExternalFilesDir(null);
//模板存放目录
File templateDir = new File(appFilesDir.getAbsolutePath() + File.separator + TemplateManage
r.TEMPLATE_LIST_DIR + File.separator + System.currentTimeMillis());
if (!templateDir.exists()) {
    templateDir.mkdirs();
}
//复制示例视频到模板目录下
```

```
//复制示例视频到模板目录下
File videoFile = new File(mOutputPath);
File videoDestFile = new File(templateDir, videoFile.getName());
FileUtils.copyFile(videoFile, videoDestFile);
Source videoSource = new Source(videoDestFile.getPath());
videoSource.setURL(AlivcResUtil.getRelationResUri(videoFile.getName()));
//复制示例封面到模板目录下
File coverFile = new File(mAliyunTemplateBuilder.getEditorProject().getCover().getPath());
File coverDestFile = new File(templateDir, coverFile.getName());
FileUtils.copyFile(coverFile, coverDestFile);
Source coverSource = new Source(coverDestFile.getPath());
coverSource.setURL(mAliyunTemplateBuilder.getEditorProject().getCover().getURL());
if (StringUtils.isEmpty(coverSource.getURL())) {
    coverSource.setURL(AlivcResUtil.getRelationResUri(coverFile.getName()));
}
//设置工程配置文件路径
File projectFile = new File(templateDir, AliyunEditorProject.PROJECT_FILENAME);
Source projectSource = new Source(projectFile.getPath());
projectSource.setURL(AlivcResUtil.getRelationResUri(AliyunEditorProject.PROJECT_FILENAME));
//构建模板
mAliyunTemplateBuilder.build(templateDir, title, videoSource, coverSource,projectSource, mT
emplateInputAdapter.list, new AliyunTemplateSourceHandleCallback() {
    @Override
    public void onHandleResourceTasks(String templateDir, List<AliyunResTask> tasks) {
        //处理模板资源
        HashMap<String, List<AliyunResTask>> map = new HashMap<>();
        //过虑重复资源
        for (AliyunResTask task : tasks) {
            if (task.getSource() == null || task.getSource().getPath() == null) {
                task.onIgnore();
                continue;
            }
            //不以alivc_resource开头需要做处理
            String url = task.getSource().getURL();
            if (StringUtils.isEmpty(url) || !url.startsWith("alivc_resource")) {
                if (map.containsKey(task.getSource().getPath())) {
                    map.get(task.getSource().getPath()).add(task);
                } else {
                    List<AliyunResTask> list = new ArrayList<>();
                    list.add(task);
                    map.put(task.getSource().getPath(), list);
                }
            } else {
                //忽略出错
                task.onIgnore();
            }
        }
        for (Map.Entry<String, List<AliyunResTask>> entry : map.entrySet()) {
            try {
                String path = entry.getKey();
                if (path == null) {
                    continue;
                }
                File srcFile = new File(path);
                File destFile = new File(templateDir, srcFile.getName());
```

```
                File destFile = new File(templateDir, srcFile.getName());
                if (!path.contains(templateDir)) {
                    FileUtils.copyFile(srcFile, destFile);
                }
                List<AliyunResTask> list = entry.getValue();
                for (AliyunResTask task:list){
                    Source source = task.getSource();
                    source.setPath(destFile.getPath());
                    source.setURL(AlivcResUtil.getRelationResUri(srcFile.getName()));
                    task.onHandleCallback(source);
                }
            } catch (Exception e) {
                //忽略出错
                List<AliyunResTask> list = entry.getValue();
                for (AliyunResTask item:list){
                    item.onIgnore();
                }
            }
        }
    }
    @Override
    public void onSuccess() {
        //成功生成模板
    }
    @Override
    public void onFailure(String msg) {
        //生成模板失败
    }
});
```

导入资源到模板生成工程配置文件

根据模板路径生成模板草稿后用于编辑，调用 `AliyunTemplateEditor` 接口编辑。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//根据模板路径生成获取模板信息
AliyunTemplate template  = AliyunTemplateFactory.getAliyunTemplate(Uri.parse(mTemplatePath)
);
//获取模板导入参数
List<AliyunTemplateParam> params = template.getImportParams();
List<AliyunClip> clips = new ArrayList<>();
//选择对应的资源替换模板参数
List<MediaInfo> data = new ArrayList<>();
for (MediaInfo mediaInfo : data) {
    if (mediaInfo.mimeType.startsWith("video")) {
        clips.add(new AliyunVideoClip.Builder()
                .source(mediaInfo.filePath)
                .startTime(mediaInfo.startTime)
                .endTime(mediaInfo.startTime + mediaInfo.duration)
                .duration(mediaInfo.duration)
                .build());
    } else if (mediaInfo.mimeType.startsWith("image")) {
        clips.add(new AliyunImageClip.Builder()
                .source(mediaInfo.filePath)
                .duration(mediaInfo.duration)
                .build());
    }
}
//根据导入视频与图片列表生成模板配置文件
AliyunEditorProject project = template.createEditorProject(TemplateMediaActivity.this, clip
s);
//把路径传给编辑器即可生成替换资源后的视频
String path = project.getProjectFile().getAbsolutePath();
```

编辑模板并合成视频

根据模板编辑器返回的 `AliyunEditorProject` 生成视频合成需要的相关参数传给 `AliyunCompose` 合成视频。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
AlivcEditOutputParam outputParam = new AlivcEditOutputParam();
//根据模板编辑器获取工程配置
AliyunEditorProject project = mAliyunTemplateEditor.getEditorProject();
//设置生成所需要的参数
outputParam.setConfigPath(project.getProjectFile().getAbsolutePath());
outputParam.setOutputVideoHeight(project.getConfig().getOutputHeight());
outputParam.setOutputVideoWidth(project.getConfig().getOutputWidth());
outputParam.setVideoRatio(((float) mPasterContainerPoint.x) / mPasterContainerPoint.y);
AliyunVideoParam param = new AliyunVideoParam.Builder()
        .frameRate(project.getConfig().getFps())
        .gop(project.getConfig().getGop())
        .crf(project.getConfig().getCrf())
        .videoQuality(VideoQuality.values()[project.getConfig().getVideoQuality()])
        .scaleMode(VideoDisplayMode.valueOf(project.getConfig().getDisplayMode()))
        .scaleRate(project.getConfig().getScale())
        .outputWidth(project.getConfig().getOutputWidth())
        .outputHeight(project.getConfig().getOutputHeight())
        .videoCodec(VideoCodecs.getInstanceByValue(project.getConfig().getVideoCodec()))
        .build();
outputParam.setVideoParam(param);
if (project.getCover() != null && !StringUtils.isEmpty(project.getCover().getPath()) && Fil
eUtils.isFileExists(project.getCover().getPath())) {
    String path = project.getCover().getPath();
    outputParam.setThumbnailPath(path);
} else {
    outputParam.setThumbnailPath(Constants.SDCardConstants.getDir(getApplicationContext())
+ File.separator + "thumbnail.jpg");
}
//编辑完成跳转到其他界面
Intent intent = new Intent();
intent.setClassName(TemplateEditorActivity.this, EditorActivity.NEXT_ACTIVITY_CLASS_NAME);
intent.putExtra(PublishActivity.KEY_PARAM_THUMBNAIL, outputParam.getThumbnailPath());
intent.putExtra(PublishActivity.KEY_PARAM_CONFIG, outputParam.getConfigPath());
intent.putExtra(PublishActivity.KEY_PARAM_VIDEO_WIDTH, outputParam.getOutputVideoWidth());
intent.putExtra(PublishActivity.KEY_PARAM_VIDEO_HEIGHT, outputParam.getOutputVideoHeight())
;
//传入视频比列
intent.putExtra(PublishActivity.KEY_PARAM_VIDEO_RATIO, outputParam.getVideoRatio());
intent.putExtra("videoParam", outputParam.getVideoParam());
//跳转到生成页面
startActivityForResult(intent, PublishActivity.REQUEST_CODE);
```

4.8. 视频拼接

短视频SDK提供了视频拼接接口AliyunIMixComposer。该接口实现离线多画面合并成一个视频的功能，例如画中画、九宫格、左右分屏、上下分屏等视频效果，支持添加多轨道视频。本文为您介绍Android端视频拼接的流程以及示例代码。

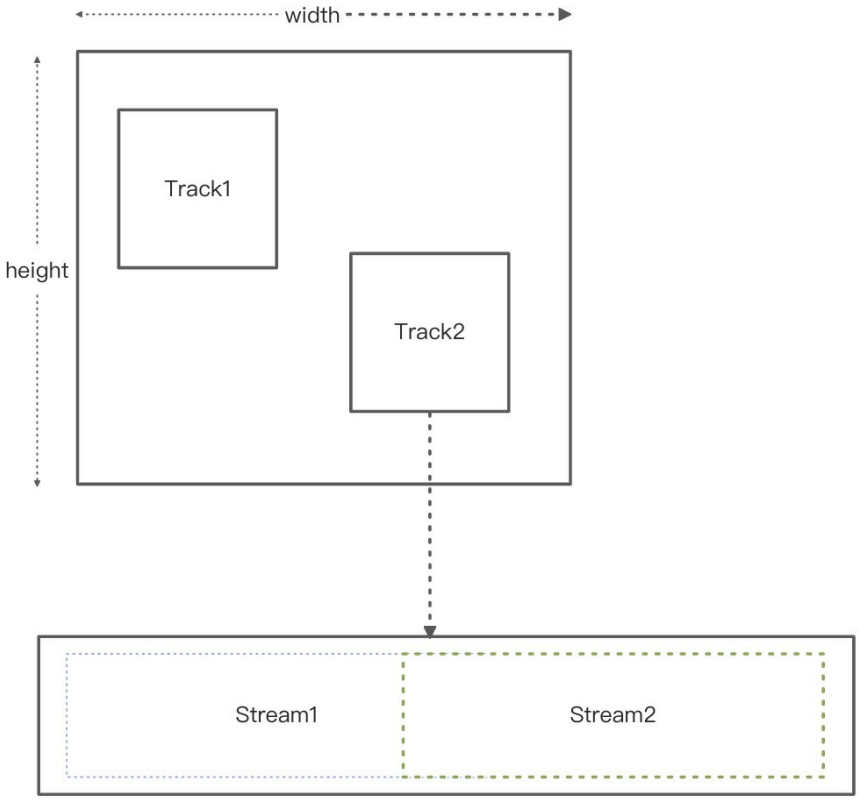
版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

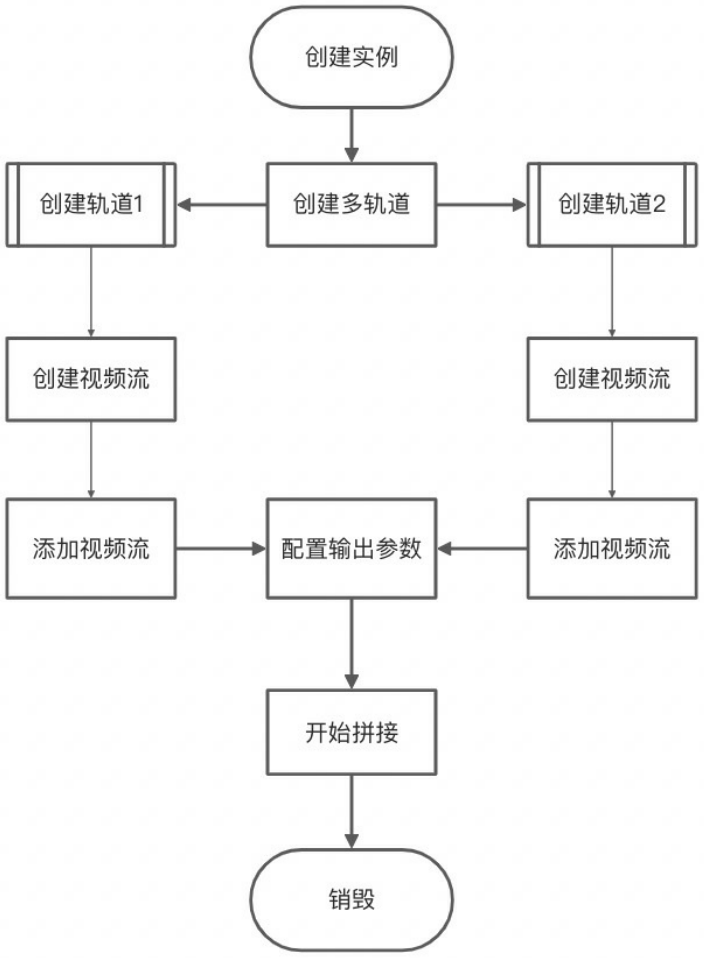
相关类功能

类名	功能
AliyunIMixComposer	拼接功能核心类，包括设置输出参数、创建轨道、添加视频流、开始拼接、设置回调等拼接核心功能。
AliyunMixComposerCreator	工厂类，创建一个AliyunIMixComposer实现类的实例。
AliyunMixTrack	视频轨道，用于在轨道上添加视频流，设置轨道的布局、音量等参数。
AliyunMixStream	视频轨道流，用于获取视频流的显示模式、文件路径、结束时间等。
AliyunMixOutputParam	拼接输出参数，设置拼接视频的宽高、输出码率、质量等级等参数。
AliyunMixCallback	拼接回调，设置拼接完成、拼接进度及拼接失败的回调。

拼接结构图



拼接流程



阶段	流程	说明	示例代码
基础	1	创建拼接实例。	创建实例
	2	创建多个轨道，创建视频流，再将视频流分别添加到各轨道上。	创建轨道
	3	配置视频拼接后的输出路径、视频宽高等参数。	配置输出参数
	4	设置回调，开始拼接。	开始拼接
	5	销毁接口，释放资源。	释放资源
进阶	6	取消、暂停、继续拼接，按需设置。	拼接控制

创建实例

创建拼接实例。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//创建实例
AliyunIMixComposer mixComposer = AliyunMixComposerCreator.createMixComposerInstance();
```

创建轨道

创建多个轨道，创建视频流，再将视频流分别添加到各轨道上。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能

```
//创建轨道
//创建轨道1
//轨道1的布局
AliyunMixTrackLayoutParam track1Layout = new AliyunMixTrackLayoutParam.Builder()
        .centerX(0.25f)
        .centerY(0.5f)
        .widthRatio(0.5f)
        .heightRatio(1.f)
        .build();
//创建轨道1实例
AliyunMixTrack track1 = mixComposer.createTrack(track1Layout);
// 创建轨道1的第一个视频流1
AliyunMixStream stream11 = new AliyunMixStream
        .Builder()
        .displayMode(VideoDisplayMode.FILL)
        .filePath("/storage/emulated/0/lesson_01.mp4")
        .streamEndTimeMills(20000)
        .build();
//添加该视频流到轨道1，注意：只能添加一个，以最后添加的为准
track1.addStream(stream11);
//创建轨道2
//轨道2参数
AliyunMixTrackLayoutParam track2Layout = new AliyunMixTrackLayoutParam.Builder()
        .centerX(0.75f)
        .centerY(0.5f)
        .widthRatio(0.5f)
        .heightRatio(1.f)
        .build();
//创建轨道2实例
AliyunMixTrack track2 = mixComposer.createTrack(track2Layout);
//创建轨道2的第一个视频流1
AliyunMixStream stream21 = new AliyunMixStream
        .Builder()
        .displayMode(VideoDisplayMode.FILL)
        .filePath("/storage/emulated/0/lesson_02.mp4")
        .streamStartTimeMills(10000)
        .streamEndTimeMills(30000)
        .build();
//添加该视频流到轨道2，注意：只能添加一个，以最后添加的为准
track2.addStream(stream21);
```

配置输出参数

配置视频拼接后的输出路径、视频宽高等参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能

```
//配置输出参数
AliyunMixOutputParam outputParam = new AliyunMixOutputParam.Builder()
        .outputPath("/sdcard/output.mp4") //拼接后的路径
        .outputAudioReferenceTrack(track2)//表示使用轨道2的音频作为最后的音频，目前音频轨道只支持一个音频流
        .outputDurationReferenceTrack(track2)//表示使用轨道2的时长作为最后输出视频的时长，如果轨道1的时长不够，则会停在最后一帧
        .crf(6)
        .videoQuality(VideoQuality.HD)
        .outputWidth(720) //视频宽度
        .outputHeight(1280) //视频高度
        .fps(30) //fps
        .gopSize(30) //gop
        .build();
mixComposer.setOutputParam(outputParam);
```

开始拼接

设置回调，开始拼接视频。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能

```
//开始拼接
AliyunMixCallback callback = new AliyunMixCallback() {
            @Override
            public void onProgress(long progress) {//拼接进度
                Log.e("MixRecord", "onProgress " + progress);
            }
            @Override
            public void onComplete() {
                Log.e("MixRecord", "onComplete"); //拼接完成
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        //该接口一定不能在回调的线程中直接调用！！！
                        //拼接完成后释放实例
                        mixComposer.release();
                    }
                });
            }
            @Override
            public void onError(int errorCode) { //拼接失败
                Log.e("MixRecord", "onError " + errorCode);
            }
};
```

释放资源

拼接完成后销毁实例，释放资源。请务必注意不要在拼接过程中销毁实例。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能

```
mixComposer.release();
```

拼接控制

按需设置取消、暂停、继续拼接。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能

```
//暂停拼接
mixComposer.pause();
//继续拼接
mixComposer.resume();
//取消拼接
mixComposer.cancel();
```

视频拼接代码示例

```
/**
 * 视频拼接Example
 */
class MixActivity : AppCompatActivity() {
    private val REQUEST_TRACK1_STREAM = 1001
    private val REQUEST_TRACK2_STREAM = 1002
    private var 示例 : AliyunIMixComposer? = null
    private lateinit var mVideoTrack1 : AliyunMixTrack
    private lateinit var mVideoTrack2 : AliyunMixTrack
    private var mVideoTrack1Duration = 0L
    private var mVideoTrack2Duration = 0L
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mix)
        findViewById<Button>(R.id.btnReset).setOnClickListener {
            findViewById<Button>(R.id.btnMix).isEnabled = false
            mMixComposer?.release()
            init()
        }
        findViewById<Button>(R.id.btnAddTrack1Stream).setOnClickListener {
            PermissionX.init(this)
                .permissions(
                    Manifest.permission.READ_EXTERNAL_STORAGE,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE
                )
                .request { allGranted, _, _ ->
                    if (allGranted) {
                        PictureSelector.create(this)
                            .openGallery(PictureMimeType.ofVideo())
                            .forResult(REQUEST_TRACK1_STREAM)
                    }
                }
        }
        findViewById<Button>(R.id.btnAddTrack2Stream).setOnClickListener {
            PermissionX.init(this)
                .permissions(
                    Manifest.permission.READ_EXTERNAL_STORAGE,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE
                )
                .request { allGranted,  ,  ->
```

```
                    if (allGranted) {
                        PictureSelector.create(this)
                            .openGallery(PictureMimeType.ofVideo())
                            .forResult(REQUEST_TRACK2_STREAM)
                    }
                }
            }
            findViewById<Button>(R.id.btnMix).setOnClickListener {
                //开始合成
                val callback: AliyunMixCallback = object : AliyunMixCallback {
                    override fun onProgress(progress: Long) { //合成进度
                        Log.e("MixActivity", "onProgress $progress")
                    }
                    override fun onComplete() {
                        Log.e("MixActivity", "onComplete")
                        ToastUtil.showToast(it.context, "视频合成成功")
                    }
                    override fun onError(errorCode: Int) {
                        Log.e("MixActivity", "onError $errorCode")
                        ToastUtil.showToast(it.context, "视频合成失败:$errorCode")
                    }
                }
                //配置输出参数
                val outputParamBuilder = AliyunMixOutputParam.Builder()
                outputParamBuilder
                    .outputPath("/storage/emulated/0/DCIM/Camera/svideo_mix_demo.mp4")
                    .crf(6)
                    .videoQuality(VideoQuality.HD)
                    .outputWidth(720)
                    .outputHeight(1280)
                    .fps(30)
                    .gopSize(30)
                if(mVideoTrack1Duration > mVideoTrack2Duration) {
                    outputParamBuilder.outputAudioReferenceTrack(mVideoTrack1)
                    outputParamBuilder.outputDurationReferenceTrack(mVideoTrack1)
                } else {
                    outputParamBuilder.outputAudioReferenceTrack(mVideoTrack2)
                    outputParamBuilder.outputDurationReferenceTrack(mVideoTrack2)
                }
                mMixComposer!!.setOutputParam(outputParamBuilder.build())
                mMixComposer!!.start(callback)
            }
            init()
        }
        private fun init() {
            mMixComposer = AliyunMixComposerCreator.createMixComposerInstance()
            //创建轨道1
            val track1Layout = AliyunMixTrackLayoutParam.Builder()
                .centerX(0.25f)
                .centerY(0.25f)
                .widthRatio(0.5f)
                .heightRatio(0.5f)
                .build()
            mVideoTrack1 = mMixComposer!!.createTrack(track1Layout)
```

```
        //创建轨道2
        val track2Layout = AliyunMixTrackLayoutParam.Builder()
            .centerX(0.75f)
            .centerY(0.75f)
            .widthRatio(0.5f)
            .heightRatio(0.5f)
            .build()
        mVideoTrack2 = mMixComposer!!.createTrack(track2Layout)
    }
    override fun onResume() {
        super.onResume()
        mMixComposer?.resume()
    }
    override fun onPause() {
        super.onPause()
        mMixComposer?.pause()
    }
    override fun onDestroy() {
        super.onDestroy()
        mMixComposer?.release()
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == Activity.RESULT_OK) {
            when (requestCode) {
                REQUEST_TRACK1_STREAM -> {
                    // onResult Callback
                    val result = PictureSelector.obtainMultipleResult(data)
                    mVideoTrack1Duration = 0
                    for (streamBean in result) {
                        // 创建轨道1的第一个视频流1
                        val stream1 = AliyunMixStream.Builder()
                            .displayMode(VideoDisplayMode.FILL)
                            .filePath(streamBean.realPath)
                            .streamStartTimeMills(mVideoTrack1Duration)
                            .streamEndTimeMills(streamBean.duration)
                            .build()
                        mVideoTrack1Duration += streamBean.duration
                        if(mVideoTrack1.addStream(stream1) == 0) {
                            ToastUtil.showToast(this, "添加轨道1视频流成功")
                        }
                    }
                }
                REQUEST_TRACK2_STREAM -> {
                    // onResult Callback
                    val result = PictureSelector.obtainMultipleResult(data)
                    mVideoTrack2Duration = 0L
                    for (streamBean in result) {
                        // 创建轨道2的第一个视频流1
                        val stream1 = AliyunMixStream.Builder()
                            .displayMode(VideoDisplayMode.FILL)
                            .filePath(streamBean.realPath)
                            .streamStartTimeMills(mVideoTrack2Duration)
                            .streamEndTimeMills(streamBean.duration)
```

```
                            .build()
                        mVideoTrack2Duration += streamBean.duration
                        if(mVideoTrack2.addStream(stream1) == 0) {
                            ToastUtil.showToast(this, "添加轨道2视频流成功")
                        }
                    }
                }
            }
        }
        if(mVideoTrack1Duration > 0 && mVideoTrack2Duration > 0) {
            findViewById<Button>(R.id.btnMix).isEnabled = true
        }
    }
}
```

XML文件配置示例

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.co
m/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"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btnReset"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="76dp"
        android:text="重置"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/btnAddTrack1Stream"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="56dp"
        android:layout_marginTop="64dp"
        android:text="添加视频1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnReset"
        app:layout_constraintEnd_toEndOf="@id/btnReset"
         />
    <Button
        android:id="@+id/btnAddTrack2Stream"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="64dp"
        android:layout_marginEnd="56dp"
        android:text="添加视频2"
        app:layout_constraintStart_toStartOf="@id/btnReset"
        app:layout_constraintTop_toBottomOf="@id/btnReset"
        app:layout_constraintEnd_toEndOf="parent"
        />
    <Button
        android:id="@+id/btnMix"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="176dp"
        android:text="开始合成"
        android:enabled="false"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

4.9. 视频上传

本文为您介绍Android端短视频SDK视频上传的功能以及流程说明。

功能介绍

视频点播支持通过多种方式上传媒体文件（音频、视频、图片等）到点播存储，详细请参见媒体上传。其中，上传SDK提供了一套单独进行合成上传的功能接口，用来实现将编辑完的视频在另一界面合成上传，核心类AliyunIVodCompose封装了视频合成与上传功能，方便客户端更好地合成与上传视频。阿里云短视频SDK通过调用核心类AliyunIVodCompose以完成视频的上传。

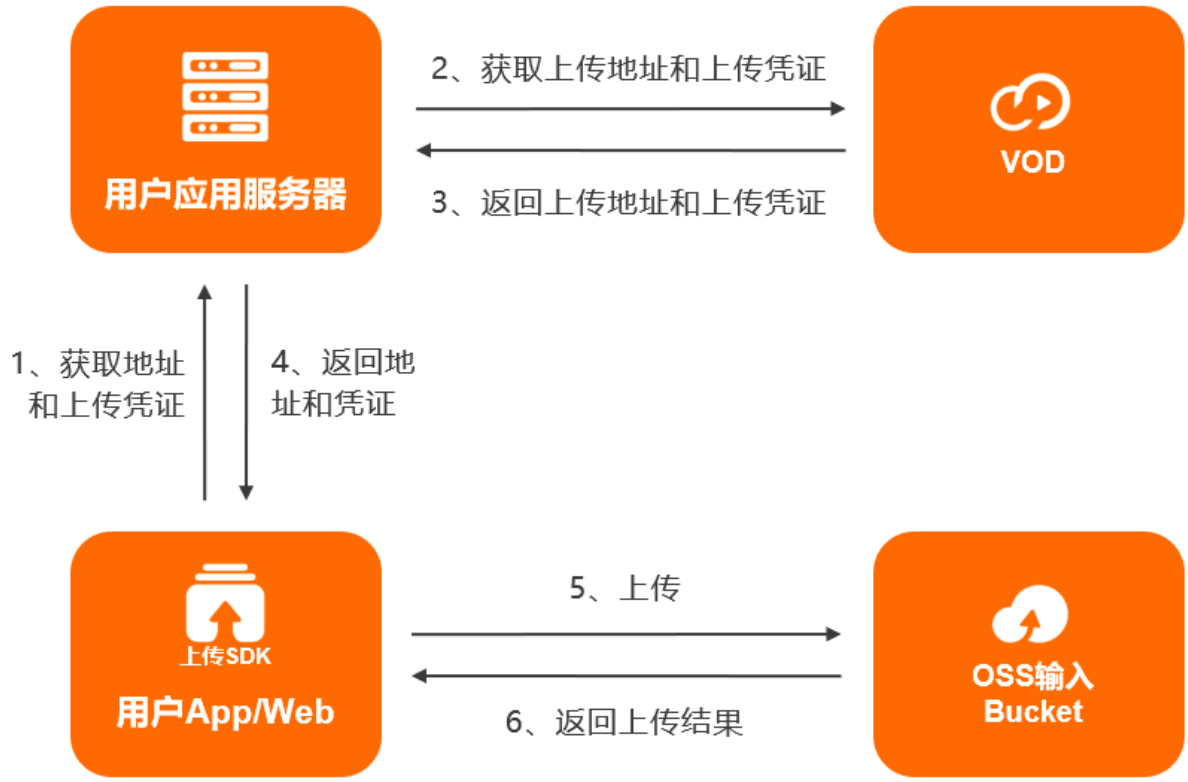
版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

相关类功能

类名	功能
AliyunIVodCompose	上传功能核心类，包括初始化、上传视频或图片、更新上传凭证等核心上传功能。
AliyunComposeFactory	工厂类。

视频上传流程



1. 当客户端（用户APP/Web端）通过上传SDK上传视频或图片时，首先向用户应用服务器发起请求，获取上传地址和凭证。
2. 应用服务器通过OpenAPI，向视频点播服务发起CreateUploadVideo（视频）或CreateUploadImage（图片）请求。
3. 请求成功，则视频点播服务将返回上传地址、上传凭证、VideoId（视频）、ImageURL（图片）给用户应用服务器。
4. 用户应用服务器返回上传地址和上传凭证给客户端。
5. 客户端添加本地文件，并设置上传地址和凭证，开始上传到OSS Bucket。
6. OSS返回上传结果。

初始化

初始化上传实例。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//创建
AliyunComposeFactory.createAliyunVodCompose();
//初始化
AliyunIVodCompose.init(Context context);
```

获取上传地址及凭证

AliyunIVodCompose是通过上传地址和上传凭证上传，在上传之前，需要获取上传地址及上传凭证，详细操作请参见获取音视频上传地址和凭证或获取图片上传地址和凭证。

> **说明** 此处只是在获取上传地址和凭证，并未开始上传，获取到上传地址和凭证后，需要再执行上传。

上传

上传视频、图片等媒体文件。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//通过上传地址和上传凭证上传视频或图片
//videoPath:视频文件地址，请与获取上传地址和凭证时输入的视频文件地址保持一致
//imagePath:图片文件地址，请与获取上传地址和凭证时输入的图片文件地址保持一致
//uploadAddress:上传地址
//uploadAuth:上传凭证，凭证过期后请参考下文刷新上传凭证
//aliyunVodUploadCallBack:上传的回调
//上传视频
AliyunIVodCompose.uploadVideoWithVod(String videoPath, String uploadAddress, String uploadA
uth, AliyunIVodUploadCallBack aliyunVodUploadCallBack);
//上传图片
AliyunIVodCompose.uploadImageWithVod(String imagePath, String uploadAddress, String uploadA
uth, AliyunIVodUploadCallBack aliyunVodUploadCallBack);
```

刷新上传凭证

因上传凭证带有时效性（UploadAuth字段的Expiration变量赋予了上传授权的过期时间，默认有效期为3000秒），所以当上传凭证过期后，需要在过期回调onUploadTokenExpired方法中重新获取上传凭证上传，更多信息，请参考刷新视频上传凭证。

```
//刷新上传凭证，uploadAuth:上传凭证
AliyunIVodCompose.refreshWithUploadAuth(String uploadAuth);
```

释放资源

上传完成后，销毁接口，释放资源。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
AliyunIVodCompose.release();
```

上传控制

上传过程中可按需暂停、继续及取消上传。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//暂停上传
AliyunIVodCompose.pauseUpload();
//继续上传
AliyunIVodCompose.resumeUpload();
//取消上传
AliyunIVodCompose.cancelUpload();
```

4.10. 视频工具

短视频SDK提供了视频文件解析、视频缩略图获取工具以辅助用户更方便的编辑录制视频等。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

视频文件解析

视频文件解析的核心类为NativeParser类，通过该类可以解析出视频宽高、格式、时长、码率、声道等信息。

获取信息通过Key的方式获取，相关参数说明如下：

Key	说明
VIDEO_STREAM_INDEX	视频流的下标。
VIDEO_CODEC	视频编码器。
VIDEO_DURATION	视频流时长。
VIDEO_FRAME_COUNT	总帧数。
VIDEO_BIT_RATE	码率（平均码率）。
VIDEO_WIDTH	宽。
VIDEO_HEIGHT	高。
VIDEO_FORMAT	视频格式。
VIDEO_GOP	关键帧间隔。
AUDIO_CODEC	音频流下标。
AUDIO_DURATION	音频流时间。

Key	说明
AUDIO_CHANNELS	音频声道。
AUDIO_BIT_RATE	音频流码率。
AUDIO_SAMPLE_RATE	音频采样率。
AUDIO_FORAMT	音频格式。

```
//1.初始化
NativeParser parser = NativeParser();

//2.解析视频
//解析视频1
parser.init(path1);
//获取信息通过Key的方式获取
String videoWidth1 = parser.getValue(NativeParser.VIDEO_WIDTH);
//解析完成后调用release
parser.release();

//解析视频2
parser.init(path2);
//获取信息通过Key的方式获取
String videoWidth2 = parser.getValue(NativeParser.VIDEO_WIDTH);
//解析完成后调用release
parser.release();

//3.销毁
parser.dispose();
```

视频缩略图

视频缩略图的核心类为AliyunIThumbnailFetcher类，通过该类可以在视频轨道上展示每一帧图片的缩略图，通常在视频编辑或裁剪时需要用到。

```
//1.创建实例
//AlivcSdkCore.register(getApplicationContext()); 创建缩略图之前，要确认SDK有进行初始化
AliyunIThumbnailFetcher fetcher = AliyunThumbnailFetcherFactory.createThumbnailFetcher();

//2.添加视频/图片源
//2.1方式一：配置文件添加，适用于SDK内操作产生的project.json
fetcher.fromConfigJson(xxxx.json);

//2.2方式二：手动添加
//添加视频
fetcher.addVideoSource(path1, startTimeMills1, endTimeMills1, overlapDurationMills1);
//添加图片
```

```
fetcher.addImageSource(path2, startTimeMills2, endTimeMills2, overlapDurationMills2);

//3.设置缩略图输出大小等信息
//cropMode
CropMode cropMode = CropMode.Mediate;// 图像裁剪方式
VideoDisplayMode videoDisplayMode = VideoDisplayMode.Scale; //图像填充方式
int cacheSize = 10; //缓存的缩略图数量
fetcher.setParameters(width, height,cropMode, videoDisplayMode, cacheSize);
//一般建议，如果是请求一系列缩略图，fastMode设置为true，会快速出图；如果是请求一张或者需要精确地时间点的缩略图，可以设置为false
fetcher.setFastMode(true);


//4.最终缩略图会在回调中给出
long[] times = {1000, 2000, 3000}; //缩略图时间戳
fetcher.requestThumbnailImage(long[] time, new AliyunIThumbnailFetcher.OnThumbnailCompletio
n() {

            @Override
            public void onThumbnailReady(Bitmap frameBitmap, long time,int index) {
                //返回的缩略图
            }

            @Override
            public void onError(int errorCode) {
                //错误码信息
            }
        });

//或者使用requestThumbnailImage(count,callback)会按照时间均分请求count张数的缩略图
fetcher.requestThumbnailImage(10, new AliyunIThumbnailFetcher.OnThumbnailCompletion() {
@Override
  public void onThumbnailReady(Bitmap frameBitmap, long time, int index) {
      // time是时间戳，对应之前传入的times里面的一个值，index是传入时间片的数组下标
 }

@Override
public void onError(int errorCode) {
    }
});
//5.用完后销毁
fetcher.release()
```

> **说明**
>
> setFastMode()接口，传入true表示快速模式，取请求时间片最近的前关键帧，可以快速获得缩略图。如果传入false表示精确模式，会精确定位到请求时间片的特定帧。以1080P，30秒的视频，取10张图为例，快速模式下大概需要1秒，精确模式下大概需要5秒。

5.iOS短视频SDK

5.1. 发布说明

V3.32.0

功能更新

- 支持Bit Code。
- 视频合拍支持实时合成。
- 视频导出支持边合成边上传。
- 修复部分已知问题。

V3.31.0

功能更新

- 字幕背景新增圆角能力。
- 编辑模式新增镜像能力。
- 优化License校验逻辑。
- 优化开启日志落地能力，使接入时排查问题更精准。
- 修复部分已知问题。

V3.30.0

功能更新

- 优化License接入方式。
- 优化裁剪性能。
- 修复部分已知问题。

V3.29.0

功能更新

- 优化裁剪性能。
- License升级，支持查看License授权信息。
- 修复部分已知问题。

V3.28.0

功能更新

修复部分已知问题。

V3.27.0

功能更新

- 新增剪同款功能。
- 多源录制支持回声消除、降噪、背景音乐与录音混音。
- 新增lut滤镜特效。

- 修复部分已知问题。

V3.26.0

功能更新

- 优化SDK的稳定性问题。
- 修复部分音频格式不支持的问题。

V3.25.1

功能更新

- 优化字幕内存。
- 修复HDR经过iPhone自带的编辑功能编辑后SDK裁剪花屏的问题。
- 修复横屏录制导出后画面显示异常的问题。
- 修复部分已知问题。

V3.25.0

功能更新

- 新增画中画功能，支持在编辑界面添加画中画。
- 新增字幕动画功能，支持对字幕（花字）等做动画。
- 草稿箱新增自定义封面图。
- 修复部分已知问题。

V3.24.0

功能更新

- 优化SDK，删除FFmpeg软编码。
- 修复在剪裁、编辑设置时，背景色不生效的问题。
- 修复部分已知问题。

V3.23.0

功能更新

- 新增草稿箱功能，支持导出草稿。
- 字幕新增背景色、对齐等功能。
- 合拍新增回声消除功能。
- Demo中美颜模块替换为Queen SDK。
- Demo中新增6个分屏滤镜特效。
- 录制支持自动删除临时视频文件。
- 支持HDR视频编辑和裁剪。

V3.22.0

功能更新

- 编辑时新增花字功能。
- 新增多种录制源合拍功能。

- 新增局部屏幕采集功能。
- 在自定义特效Shader类中新增时间的内建变量（BUILTIN_PROGRESS）。
- 提升SDK稳定性，修复部分场景下不稳定的问题。

接口变动

- 新增接口：

  (AliyunStickerManager *)getStickerManager;

- 废弃接口：
  - (AliyunPasterManager *)getPasterManager;
  - (id<AliyunIPasterRender>)getPasterRender;

V3.21.0

功能更新

- 合拍功能支持调整视图层次以及设置边框效果。
- 支持镜像视频导入编辑。
- 优化软编译时的内存问题。
- 修复SDK稳定性问题。

V3.20.0

功能更新

- 编辑模块音频增加淡入淡出效果。
- 编辑模块增加组合字幕功能。
- 编辑模块增加基础编辑能力。
- 获取视频特定时间戳缩略图能力。
- 修复编辑场景视频导出帧率设置不生效的问题。
- 修复SDK稳定性问题。

V3.19.0

功能更新

- 新增编辑模块音频降噪功能。
- 新增合拍和视频拼接功能，支持设置背景图片和背景颜色。
- 新增合拍和视频合并功能，支持音轨合并。
- 新增合拍增加回声消除能力。
- 修复设置水印、图片，添加某些透明光晕图片，光晕变色的问题。
- 修复添加静态图片旋转角度不对问题。

接口变动

废弃接口

AliyunCamera & AliyunIRecorder，@property(nonatomic, assign) BOOL useAudioSessionModeVideoRecording;

V3.18.1

功能更新

修复iOS部分机型硬编码内存问题。

V3.18.0

功能更新

增加合拍视频指定使用的音轨功能（视频原音、录制声音、静音）。

V3.17.1

功能更新

- 前置摄像头支持画面缩放。
- 修复某些机型合成后opengl导致的闪退问题。

V3.17.0

功能更新

- 优化萝莉音效、新增方言音效。
- 修复裁剪和编辑模块导入iPhone 12自带相机拍摄的视频后出现的绿屏问题。

V3.16.2

功能更新

修复高斯模糊背景问题。

V3.16.1

功能更新

修复录制时长大于视频时长时不能合成问题。

V3.16.0

功能更新

- 恢复主流动画功能。
- 修复线上反馈偶现崩溃问题。
- 修复长视频可能出现的播放卡顿问题。
- 修复横屏录制水印方向问题。

V3.15.0

功能更新

- 修复合成视频播放卡顿问题。
- 修复多段视频变速失效问题。
- 新增基于自定义特效制作规范的两组转场、滤镜效果转场与滤镜效果。

接口变动

- 新增自定义特效参数调节接口，支持实时调节特效参数。
- 支持自定义滤镜、转场特效，自定义特效制作规范请参考官方文档。

V3.14.0

功能更新

- 优化裁剪模块，保证反复裁剪颜色不失真。
- 优化录制实现，针对退后台、硬件资源占用等异常情况提升稳定性。
- 修复已知几处内存泄漏并优化部分性能。

问题修复

- 修复录制过程中退后台偶现卡死问题。
- 修复录制过程中音频资源被占用导致异常的问题。
- 修复编辑设置背景颜色不生效问题。
- 修复iOS编辑功能变化视图后播放画面局部放大问题。
- 修复部分内存泄漏和其他已知问题。

接口变动

AliyunVideoSDKPro.framework（动态库）拆分为AliyunVideoSDKPro.framework（静态库）和AliyunVideoCore.framework（动态库）两部分。如果您采用手动方式集成SDK。

详情请参考集成SDK。

V3.13.0

功能更新

- 录制模块稳定性，性能全面优化。
- 录制模块支持基于RACE的美颜美型功能。
- 编辑模块提升H265视频流畅性。

接口变动

录制模块废弃mv接口，去除添加mv功能。

V3.12.0

功能更新

- 增加日志分析功能提供开关日志分析的接口[AliyunVideoSDKInfo setDebugLogLevel:]提供三个等级供用户设置：

```
AlivcDebugLogClose 关闭日志分析功能
AlivcDebugLogNormal 能分析warning，error级别的日志，建议使用这个等级来做日志分析
AlivcDebugLogAll 全量log分析，只建议在定位疑难问题时开启，不建议在正式发版中使用以上功能只会做SDK的日志分析
```

- 编辑模块性能提升。

问题修复

录制模块修复线程未退出问题。

接口变动

编辑模块废除applyRunningDisplayMode接口，去除动态切换内容模式功能。

V3.11.0

功能更新

- 提升片段录制起停的速度和录制合成的速度，分段录制更加流畅。
- 优化录制进度回调粒度和精准度。
- 精准控制gop，提升部分场景下的转码速度。
- 优化切换摄像头耗时。

问题修复

- 修复gif帧数解析不准确的问题。
- 修复特定视频倒播开始播放时卡顿问题。
- 修复录制生成视频时长不精准的问题。
- 修复多段录制输出视频音视频不同步。

接口变动

- 对外错误码统一所有错误码统一到AliyunVideoCoreError。
- 新增方法：NSString* AlivcErrorMessage(int code)，获取错误描述。

V3.10.5

功能更新

- 新增录制合拍功能（AliyunMixRecorder)。
- 新增多轨道视频拼接（可以实现画中画，左右分屏等效果）(AliyunMixComposer)。

V3.10.0

功能更新

- 编辑新增大魔王，小黄人音效。
- 编辑新增mjpeg视频格式支持。
- 编辑播放提升对部分损坏视频文件的兼容性。
- 编辑新增draw方法支持强制绘制一帧。

问题修复

- 录制修复小段录制视频时长不准确问题。
- 录制修复添加水印后退后台水印消失的问题。
- 录制修复切换前后摄像头卡顿的问题。
- 编辑修复合成导出退后台可能导致崩溃的问题。

V3.9.0

功能更新

- 新增音效接口，提供萝莉，大叔，混响，回声四种音效。
- 提升编辑模块seek性能。
- 提升SDK稳定性。

V3.8.0

功能更新

- 提升录制稳定性。

- 提升了选择裁剪时间的准确性。
- 优化了编辑播放能力，流畅播放不卡顿。
- 优化了编辑合成的速度。
- 修复了一些bug。
- 合成支持退后台返回继续合成。
- 考虑到SDK稳定性监控和未来数据相关需求，短视频目前必须要依赖库：手动集成需要添加AlivcConan.framework，cocoaPods集成 'pod 'AlivcConan', '0.9.0' 可以参考demo。

V3.7.7

功能更新

- SDK稳定性提升。
- 裁剪，合成清晰度提升。

V3.7.5

功能更新

- 修复导入iOS12系统生成的HEVC视频倒播显示异常的bug。
- 修复编辑使用第三方渲染接口可能导致crash的bug。
- 时间特效播放流畅度提升。
- 导入GIF图片适配性增强。
- 支持导入奇数分辨率视频。
- 优化了多段录制音视频同步。
- 提升稳定性。

V3.7.0

功能更新

- 新增转场功能，包含淡入淡出、多边形转场、百叶窗等主流效果。
- 新增特技效果接口，包含基本动画（旋转、平移、缩放、透明度）和自定义动画效果（线性擦除）。
- 新增指定流、指定时间段添加高斯模糊效果。
- 新增指定流、指定时间段添加显示模式（填充、裁剪）。
- 新增配音接口，支持多段配音和变速。
- 新增多段视频加多个变速时间特效（反复和倒放还是只支持单段视频的）。

接口变动

- 编辑预览播放结束后需要调用replay接口，以前调用play接口。
- 编辑新增prepare预加载数据接口，此接口在startEdit之后调用。
- 水印的位置大小的参考坐标修改为输出分辨率坐标。
- 在调用转场效果时，需要先调用stopEdit,然后调用转场，再调用startEdit。
- AliyunPasterController删除delegate属性。
- AliyunEditor删除destroyAllEffect接口。
- QuCore-ThirdParty.framework改为alivcffmpeg.framework。

V3.6.5.5

功能更新

兼容Xcode10.x。

V3.6.5.3

功能更新

- 修复偶现录制添加mv锁屏导致的crash。
- 修复倒播加特效滤镜时间不准确。
- 修复部分视频色域显示不准确。
- 支持aac sbr格式音频。

V3.6.5

功能更新

- 修复偶现导出crash的bug。
- 提升视频倒播流畅度。

V3.6.0

功能更新

包大小和基本问题

包名称	包大小（单位M）
AliyunVideoSDKPro.framework3.5.0 release	4.9M
AliyunVideoSDKPro.framework3.5.0 debug	10.1M
AliyunVideoSDKPro.framework3.6.0 release	7.6M
AliyunVideoSDKPro.framework3.6.0 debug	15.7M
QuCore-ThirdParty.framework3.5.0 release	9.3M
QuCore-ThirdParty.framework3.5.0 debug	23.1M
QuCore-ThirdParty.framework3.6.0 release	10.2M
QuCore-ThirdParty.framework3.6.0 debug	23.2M

说明 需要同时替换AliyunVideoSDKPro.framework和QuCore-ThirdParty.framework 否则会产生异常（如合成crash等）

接口变动

- 水印接口

  -(void)setWaterMark: frame接口废弃，启用-(void)setWaterMark示例：

```
NSString watermarkPath = [[NSBundle mainBundle] pathForResource:@”watermark” ofType:@”png
”];
AliyunEffectImage effectImage = [[AliyunEffectImage alloc] init];
effectImage.frame = CGRectMake(10, 10, 28, 20);
effectImage.path = watermarkPath;
[self.editor setWaterMark:effectImag];
```

- 片尾接口

  3.6支持预览时查看需要设置-(void)setTailWaterMark，示例：

```
NSString tailWatermarkPath = [[NSBundle mainBundle] pathForResource:@”tail” ofType:@”png”
];
AliyunEffectImage tailWatermark = [[AliyunEffectImage alloc] initWithFile:tailWatermarkPa
th];
tailWatermark.frame = CGRectMake(CGRectGetMidX(self.movieView.bounds) - 84 / 2, CGRectGet
MidY(self.movieView.bounds) - 60 / 2, 84, 60);
tailWatermark.endTime = 2;
[self.editor setTailWaterMark:tailWatermark];
```

- 添加音乐接口
  - 3.6版本支持多路音频混流，如果只需要加一路音频，需要调用remove接口示例：

```
AliyunEffectMusic *music = [[AliyunEffectMusic alloc] initWithFile:path];
[self.editor removeMusics];//只要一路音频的情况下可以调用
[self.editor applyMusic:music];
```

  - 在加音乐的时候需要调用 -（void）removeMVMusic 如：AliyunEffectMusic *music = [[AliyunEffectMusic alloc] initWithFile:path]。

```
[self.editor removeMVMusic];
[self.editor removeMusics];
[self.editor applyMusic:music];
```

  - 3.6版本支持音乐选取自身的时间段进行播放如：

```
AliyunEffectMusic music = [[AliyunEffectMusic alloc] initWithFile:path];
music.startTime = startTime; //音乐本身的开始播放时间
music.duration = duration; //音乐本身的播放持续时间
music.streamStartTime = streamStart [_player getStreamDuration]; //音乐在播放时间轴上的开
始播放时间
music.streamDuration = streamDuration * [_player getStreamDuration]; //音乐在播放时间轴上
的持续播放时间
```

- 新增时间特效显示。
  - Demo中接口-(void)addTimelineTimeFilterItem，具体代码参见demo。

- 时间特效和特效滤镜的相互影响

  例如在全程变速的情况下加特效滤镜或者倒播的情况下加滤镜这个交互需要注意添加的时间段显示是否正确这部分的代码集中在3.6的：

```
(void)didBeganLongPressEffectFilter:(AliyunEffectFilterInfo *)animtinoFilterInfo ;
(void)didTouchingProgress;
(void)didEndLongPress;
```

  三个函数中，用户可以直接参考代码：

```
AliyunEffectFilter *animationFilter = [[AliyunEffectFilter alloc] initWithFile:[animtin
oFilterInfo localFilterResourcePath]];
float currentSec = [self.player getCurrentTime];
float currentStreamSec = [self.player getCurrentStreamTime];
animationFilter.startTime = currentSec;
animationFilter.endTime = [self.player getDuration];
animationFilter.streamStartTime = currentStreamSec;  //新增 streamStartTime 如果在有时间特效的情况下需设置
animationFilter.streamEndTime = [self.player getStreamDuration];//新增 如果在有时间特效的情况下需设置
[self.editor applyAnimationFilter:animationFilter];
```

  为了兼容老版本，如果在没有时间特效的情况下，仍然可以直接设置startTime和endTime，无需设置streamStartTime和streamEndTime。

- 新增时间特效接口

  接入到3.5不会产生兼容问题具体代码参见3.6Demo示例：

```
AliyunEffectTimeFilter *timeFilter = [[AliyunEffectTimeFilter alloc] init];
timeFilter.startTime = [_player getCurrentStreamTime];
timeFilter.endTime = timeFilter.startTime + 1;
timeFilter.type = TimeFilterTypeSpeed;
timeFilter.param = 0.5;
[self.editor applyTimeFilter:timeFilter];
```

- 播放状态和接口调用的调整（如前后台切换，页面切换）

  相比较3.5版本，3.6版本在内部对前后台切换和页面切换进行了处理，具体表现在：

  - -(void)setActive弃用。
  - viewWillAppear与viewWillDisappear的处理：viewWillDisappear时不需要stopEdit销毁整个AliyunEdit，只需要调用stop停止播放。同理，viewWillAppear时，只需要调用play重新播放。
  - 退后台与返回前台的处理：

    退后台，SDK会停止播放或导出，回前台默认会重新播放（或暂停）

    发生错误处理：

    3.6版本播放或导出发生错误时，会停止播放或导出，同时通过playError或exportError返回，用户可以根据情况处理。

- AliyunImporter类接口调整，以下三个接口废弃（兼容老版本，调用暂时不会产生问题）。
  - (void)addVideoWithPath:(NSString *)videoPath animDuration:(CGFloat)animDuration。
  - (void)addVideoWithPath:(NSString *)videoPath startTime:(CGFloat)startTime duration:(CGFloat)duration animDuration:(CGFloat)animDuration。

- (NSString)addImage:(UIImage)image duration:(CGFloat)duration animDuration:(CGFloat)animDuration。

新的接口通过构建AliyunClip对象，调用 -(void)addMediaClip:(AliyunClip*)clip以添加视频片段示例：

```
AliyunImporter *importor = [[AliyunImporter alloc] initWithPath:root outputSize:_compositionConfig.outputSize];
AliyunClip *clip = [[AliyunClip alloc] initWithVideoPath:info.sourcePath startTime:info.startTime duration:info.duration animDuration:i == 0 ? 0 : 1];
[importor addMediaClip:clip];
```

说明 由于有时间特效，播放器player有几个基本概念需要区别：

- /* 获取总时长，单位：秒 @return 总时长 */
  (double)getDuration
- /* 获取当前播放时间，单位：秒 */
  (double)getCurrentTime
- /* 获取原始视频流时长，单位：秒 @return 总时长 */
  (double)getStreamDuration
- /* 获取原始视频流播放时间，单位：秒 */
  (double)getCurrentStreamTime

举例

- 例如：一个视频原本是15s，加全程快速2倍播放。那么，从效果上看这个视频会被按照2倍去快速播放，此时，getDuration为7.5s。getCurrentTime为当前播放的时间（假如为3.5s），getStreamDuration为15s。getCurrentStreamTime为7s（在getCurrentTime为3.5s的情况下）。
- 再比如，原本视频15s，全程慢速2倍播放，那么getDuration为30s。getCurrentTime假定为10s，则getStreamDuration为15s。getCurrentStreamTime为5s。
- 再再比如，原本视频为15s，全程倒播，那么getDuration为15s，getCurrentTime假定为6s，则getStreamDuration为15s，getCurrentStreamTime为9s。

以上为全程的情况下，那么在局部（如局部变速和反复）均按照以上的规则进行计算换算。

其他

ReleaseNote

- 新增时间特效功能接口
  - (int)applyTimeFilter:(AliyunEffectTimeFilter *)filter。
  - (int)removeTimeFilter。
- AliyunImporter类接口进行调整废弃原本以下三个接口
  - (void)addVideoWithPath:(NSString *)videoPath animDuration:(CGFloat)animDuration。
  - (void)addVideoWithPath:(NSString *)videoPath startTime:(CGFloat)startTime duration:(CGFloat)duration animDuration:(CGFloat)animDuration。

- (NSString)addImage:(UIImage)image duration:(CGFloat)duration animDuration:(CGFloat)animDuration

  新的接口通过构建 AliyunClip对象，调用 -(void)addMediaClip:(AliyunClip*)clip以添加视频片段示例：

```
AliyunImporter importor = [[AliyunImporter alloc] initWithPath:root outputSize:_compositionConfig.outputSize];
AliyunClip *clip = [[AliyunClip alloc] initWithVideoPath:info.sourcePath startTime:info.startTime duration:info.duration animDuration:i == 0 ? 0 : 1];
[importor addMediaClip:clip];
```

- 播放状态和接口调用的调整相比较3.5版本，3.6版本在内部对前后台切换和页面切换进行了处理，具体表现在：

  (void)setActive弃用。

  viewWillAppear与viewWillDisappear的处理：viewWillDisappear时不需要stopEdit销毁整个AliyunEdit，只需要调用stop停止播放。同理，viewWillAppear时，只需要调用play重新播放。

  退后台与返回前台的处理：退后台SDK会停止播放或导出，回前台默认会重新播放（或暂停）。

  发生错误处理：3.6版本播放或导出发生错误时，会停止播放或导出，同时通过playError或exportError返回，用户可以根据情况处理。

- 播放器新增接口
  - (double)getStreamDuration; //获取原始视频流时长，单位：秒。
  - (double)getCurrentStreamTime; //获取原始视频流播放时间，单位：秒。
- 水印
  - 废弃接口：(void)setWaterMark: frame。
  - 新增接口：(void)setWaterMark:(AliyunEffect*)waterMark。
  - 片尾水印支持预览，需调用-(void)setTailWaterMark接口。
- 音乐接口
  - 支持多路音频流，支持选择音频时间段播放。如果只需要加一路音频，需要调用remove接口。示例：

```
AliyunEffectMusic *music = [[AliyunEffectMusic alloc] initWithFile:path];
[self.editor removeMusics];//只要一路音频的情况下可以调用
[self.editor applyMusic:music];
```

  - 支持音乐选取自身的时间段进行播放。示例：

```
AliyunEffectMusic music = [[AliyunEffectMusic alloc] initWithFile:path];
music.startTime = startTime; //音乐本身的开始播放时间
music.duration = duration; //音乐本身的播放持续时间
music.streamStartTime = streamStart [_player getStreamDuration]; //音乐在播放时间轴上的开始播放时间
music.streamDuration = streamDuration * [_player getStreamDuration]; //音乐在播放时间轴上的持续播放时间
```

5.2. 集成SDK

本文为您介绍iOS短视频SDK的pod集成和手动配置操作。

前提条件

开发前的环境要求如下表所示。

类别	说明
系统版本	支持iOS 9.0及以上版本。
macOS High Sierra版本	支持macOS High Sierra 10.13及以上版本。
Xcode版本	支持Xcode 9.0及以上版本，下载Xcode。

背景信息

短视频SDK分为专业版，标准版和基础版，三个版本使用的库名分别为专业版（AliyunVideoSDKPro.framework），标准版（AliyunVideoSDKPro.framework）和基础版（AliyunVideoSDKBasic.framework）。

- 基础版只包含录制与裁剪模块。
- 专业版和标准版包含全功能模块，标准版的高级接口需要单独授权才能调用。

> **说明** 短视频SDK的功能更新，请参见发布说明。

pod方式集成（推荐）

操作步骤

1. 在Podfile文件中添加依赖，各版本依赖如下表所示。

   > **注意** 如果使用短视频SDK 3.24.0及以上版本，请务必使用FFmpeg 4.3.0及以上版本。

版本	Podfile文件对应依赖
专业版	`pod 'AliyunVideoSDKPro', '~> 3.32.0'`
标准版	`pod 'AliyunVideoSDKStd', '~> 3.32.0'`
基础版	`pod 'AliyunVideoSDKBasic', '~> 3.32.0'`

2. 更新pod repo。

   ```
   pod repo update
   ```

3. 安装pod。

   ```
   pod install
   ```

说明 请确保网络环境可以访问更新pod仓库，并在pod install安装完成后检查framework版本号是否和官网最新版一致。

手动方式集成（不推荐）

专业版、标准版本需要手动方式集成需要从Github下载最新的release包，下载相应版本的6个库和一个bundle资源包。

名称	类型	描述	下载地址
AliyunVideoSDKPro	静态库	短视频SDK	专业版/标准版
AliyunVideoAIO	动态库	短视频SDK	专业版/标准版
AliyunVideoSDKPro.bundle	资源包	短视频SDK	专业版（仅专业版需要资源包）
alivcffmpeg	动态库	ffmpeg库	Github下载地址
AlivcConan	动态库	工具库	Github下载地址
VODUpload	静态库	点播上传库	Github下载地址
AliyunOSSiOS	静态库	OSS上传库	Github下载地址

基础版本需要从Github下载最新的release包，下载相应版本的4个库。

名称	类型	描述	下载地址
AliyunVideoSDKBasic	静态库	短视频SDK	基础版
AliyunVideoAIO	动态库	短视频SDK	基础版
alivcffmpeg	动态库	ffmpeg库	Github下载地址
AlivcConan	动态库	工具库	Github下载地址

操作步骤

注意

1. 动态库需要在TARGETS > General > Frameworks, Libraries, and Embedded Content中导入添加。
2. 提交到App Store需要剥离模拟器版本（x86架构）的动态库，否则会被拒绝。剥离的方式有两种：
    - 使用命令行工具lipo剥离模拟器架构的framework。
    - 通过pod方式集成，pod会在打包时自动剥离掉模拟器架构的framework。

1. 添加动态库。

   单击**General**，选择**Frameworks, Libraries, and Embedded Content**。单击**+**，然后单击**Add Other...**，分别导入AliyunVideoAIO.framework， AlivcConan.framework和alivcffmpeg.framework依

赖。导入之后请选择Embed &Sign。

2. 添加静态库。

   单击General，选择Frameworks，Libraries和Embedded Content。单击+，然后单击Add Other...，分别导入AliyunVideoSDKPro.framework， AliyunOSSiOS.framework和VODUpload.framework依赖。

3. 添加其他库依赖。

   单击General，选择Frameworks, Libraries, and Embedded Content。单击+，分别添加MobileCoreServices.framework，SystemConfiguration.framework和libresolv.tbd依赖。

4. 专业版SDK需要将AliyunVideoSDKPro.bundle依赖导入到工程，基础版和标准版SDK可忽略此步骤。

   单击Build Phases，选择Copy Bundle Resources，单击+，然后单击Add Other...，导入AliyunVideoSDKPro.bundle。

> 说明
>
> 手动方式集成较为繁琐，同时需要注意动态库提交App Store问题，推荐使用pod方式集成。
>
> 编译报错时，请修改Build Settings > Apple Clang - Language > Compile Sources As，修改为Objective-C++。

配置项目

SDK集成后，打开项目工程并修改以下配置：

1. 配置Build Setting > Linking > Other Linker Flags，添加-ObjC。
2. 配置Build Setting > Build Options > Enable Bitcde，设为NO。
3. 打开工程info.Plist，添加以下权限：

```
Privacy - Camera Usage Description
Privacy - Microphone Usage Description
Privacy - Photo Library Usage Description
```

配置License

获取到License后，需要按以下操作配置License文件。License的获取及详细信息请参见获取短视频SDK License。

把下载的License文件导入到App工程中，在Info.plist文件中添加两个key，第一个key为AlivcLicenseKey，value为LicenseKey的值；第二个key为AlivcLicenseFile，value为内置证书文件（相对于mainBundle）的路径。示例如下所示：

key	value
AlivcLicenseKey	LicenseKey的值。取值示例：MoCTfuQ391Z01mNqG8f8745e23c8a457a8ff8d5faedc1****
AlivcLicenseFile	内置证书文件（相对于mainBundle）的路径。

相关文档

API接口信息，请参见API参考。

后续步骤，请参见初始化SDK。

5.3. 初始化SDK

前提条件

初始化SDK前需要完成以下操作：

- 获取并开通短视频SDK License，详细内容请参见获取短视频SDK License。
- 集成短视频SDK。从3.29.0版本开始，注册SDK之前，需要先配置License，详细内容请参见集成SDK。

引入头文件

SDK头文件主要说明使用短视频SDK时所需要遵循的接口使用规范，使用短视频SDK前需要先引入头文件。

- 专业版&标准版

```
#import <AliyunVideoSDKPro/AliyunVideoSDKPro.h>
```

- 基础版

```
#import <AliyunVideoSDKBasic/AliyunVideoSDKBasic.h>
```

注册SDK

从3.29.0版本开始，短视频SDK升级了License服务，在App启动后必须进行注册，否则无法使用短视频SDK的功能。具体如下：

```
// ≥ 3.30.0版本：
NSError *error = [AliyunVideoSDKInfo registerSDK]; // 返回error为nil表示注册成功。
// 因为注册失败基本属于接入错误，所以建议直接加上Assert就可以在接入调试时显示错误和修复建议。
NSAssert2(error == nil, @"注册SDK失败！%@；%@", error.localizedDescription, error.localizedRecoverySuggestion);
// 3.29.0版本：
// 请把获取到的LicenseFile改名为license.crt，加到您的App工程里，通过以下方式获取LicenseFile路径。
NSString *licenseFilePath = [NSBundle.mainBundle pathForResource:@"license" ofType:@"crt"];
// 使用获取到的LicenseKey和LicenseFilePath注册SDK。
[AliyunVideoSDKInfo registerSDKWithLicenseKey:LicenseKey licenseFile:licenseFilePath];
```

> **注意** 注册成功代表License已经注入到SDK内部，但是不代表鉴权结果成功。

您可以通过代码查询当前License状态信息。

```
AliyunVideoLicense *license = AliyunVideoLicenseManager.CurrentLicense;
```

在您使用具体功能或增值服务时，SDK内部会发生鉴权，如果鉴权失败会在对应的接口返回结果。您也可以在统一的地方监听鉴权结果。

```
// ≥ 3.30.0版本
AliyunVideoLicenseManager.EventDelegate = self; // 具体请参考AliyunVideoLicenseEventDelegate协议说明
```

您可以通过代码主动查询鉴权结果。

```
AliyunVideoLicenseResultCode code = [AliyunVideoLicenseManager check];
```

如果您续费了或者购买了增值服务也可以主动更新License（默认每15分钟检查更新一次）。

```
[AliyunVideoLicenseManager Refresh:^(AliyunVideoLicenseRefreshCode code){
    // 更新结果: code
}];
```

日志输出

默认情况下，短视频SDK仅输出警告和出错日志（AlivcLogWarn）。用户可以设置日志输出级别以打印更多日志作为故障排查参考。

```
// 日志级别定义如下:
typedef NS_ENUM(NSInteger, AlivcLogLevel)
{
    AlivcLogClose = 1,
    AlivcLogVerbose,
    AlivcLogDebug,
    AlivcLogInfo,
    AlivcLogWarn,
    AlivcLogError,
    AlivcLogFatal
};
// 设置日志输出级别
[AliyunVideoSDKInfo setLogLevel:AlivcLogDebug];
```

版本信息

您可以通过以下方式打印版本信息，以确保引入的SDK版本正确或帮助排查SDK使用过程中的问题。

```
[AliyunVideoSDKInfo printSDKInfo];
```

您还可以打印所需要的信息。

```
NSString *version = [AliyunVideoSDKInfo version]; // 版本号
NSString *module = [AliyunVideoSDKInfo module];   // 版本类型
int moduleCode =[AliyunVideoSDKInfo versionCode]; // 版本类型代码
NSString *buildId =[AliyunVideoSDKInfo videoSDKBuildId]; // 版本打包ID
NSLog(@"\n==============\nVERSION: %@\nBUILD_ID: %@\nMODULE: %@\nMODULE_CODE: %d\n==========
====", version, buildId, module, moduleCode);
```

5.4. 视频录制

5.4.1. 概述

本文介绍视频录制的类型及版本差异。

录制类型

视频录制分为三种类型：基础录制、视频合拍、多源录制。

- 基础录制：摄像头采集录制功能。
- 视频合拍：使用一个已有视频作为样本视频，与摄像头采集的数据按照特定的布局方式进行合拍录制。
- 多源录制：支持View录制、摄像头录制等多种视频采集源按需组合的合拍录制。

版本支持

各版本支持的录制类型如下表所示，表格中的✓表示支持，×表示不支持。

录制类型	基础版	标准版	专业版
基础录制	✓	✓	✓
视频合拍	×	✓	✓
多源录制	×	✓	✓

5.4.2. 基础录制

短视频SDK提供基础视频录制，同时支持添加配乐，变速录制，人脸贴纸等录制效果。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

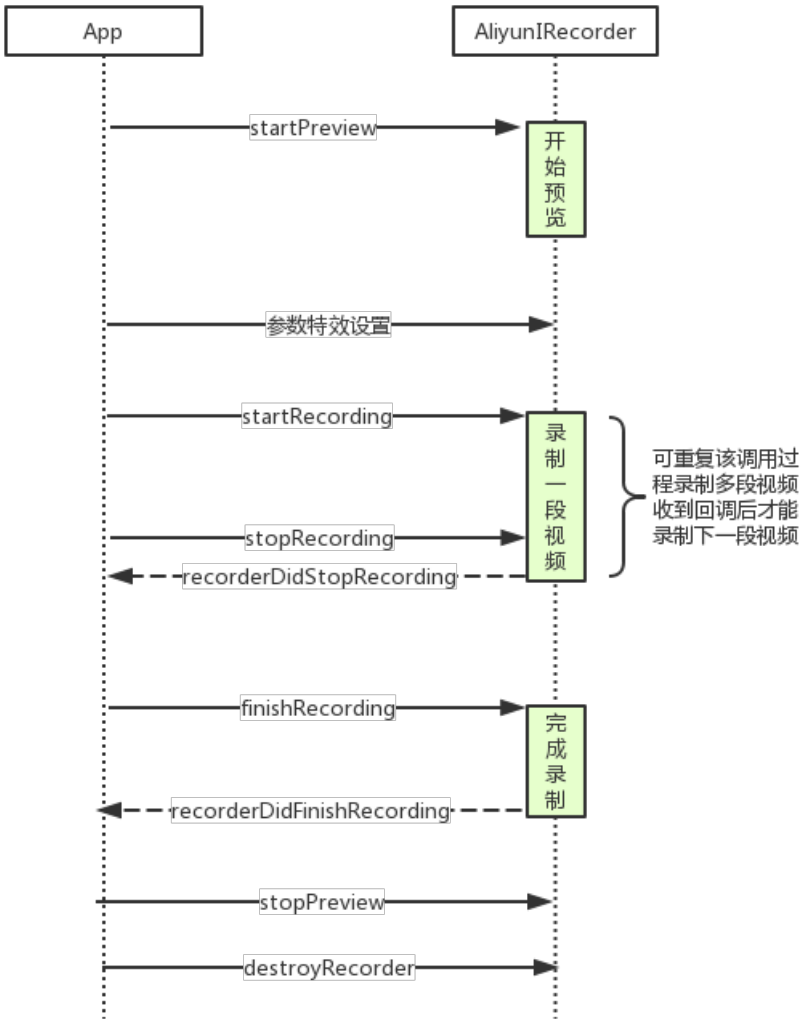
相关类功能

名称	功能
AliyunIRecorder	录制功能核心类，包括录制、设置预览、设置特效、设置回调等核心录制功能。
AliyunClipManager	录制片段管理器，获取片段信息，对视频片段进行删除操作等。
AliyunIRecorderDelegate	录制代理回调。

录制流程

> **说明** 录制功能需要获取摄像头和麦克风权限，否则无法录制。

基础录制的流程如下所示：



阶段	流程	说明	代码示例
基础	1	初始化AliyunIRecorder类，创建录制接口，并配置录制参数。	初始化
	2	创建开始预览和结束预览。	预览控制
	3	创建开始录制片段、取消录制片段、停止录制片段。	开始录制
	4	创建结束录制相关信息。	结束录制
	5	配置录制最大或最小时长、删除录制片段、获取录制片段数量等，可按需配置。	录制片段管理

阶段	流程	说明	代码示例
进阶	6	配置美颜、滤镜、背景音乐等录制特效，可按需配置。	设置特效
	7	对于常见的回调事件处理及对锁屏、来电、退后台等事件的特殊处理。	事件处理

初始化

初始化AliyunIRecorder类，创建录制接口，并配置录制参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

> 说明
> - taskPath表示文件夹路径，用来存放录制相关配置。
> - preview长宽比和videoSize保持一致。

```
CGSize resolution = CGSizeMake(720, 1280);           // 720P
AliyunIRecorder *recorder = [[AliyunIRecorder alloc] initWithDelegate:self videoSize:resolution];
recorder.taskPath = taskPath;//设置文件夹路径
recorder.preview = self.videoView;//设置预览视图
recorder.recordFps = 30;
recorder.GOP = recorder.recordFps * 3;  // 预计隔3s一个关键帧
recorder.outputPath = [taskPath stringByAppendingPathComponent:@"output.mp4"];//设置录制视频输出路径
recorder.frontCaptureSessionPreset = AVCaptureSessionPreset1280x720;
recorder.backCaptureSessionPreset = AVCaptureSessionPreset1280x720;
// 录制片段设置
recorder.clipManager.deleteVideoClipsOnExit = YES; // 退出时自动删除所有片段，也可以考虑在拍摄结束后自行删除taskPath
recorder.clipManager.maxDuration = 15;
recorder.clipManager.minDuration = 3;
self.aliyunRecorder = recorder;
```

预览控制

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

开启预览

```
// AliyunIRecorderCameraPositionFront 前摄像头启动预览
// AliyunIRecorderCameraPositionBack 后摄像头启动预览
[self.aliyunRecorder startPreviewWithPositon:AliyunIRecorderCameraPositionFront];
```

调整预览参数

开启预览后，可按需对相关预览参数进行调整。

```
// 开关手电筒，为后摄像头时开启
AliyunIRecorderTorchMode torchMode = self.aliyunRecorder.cameraPosition == AliyunIRecorderC
ameraPositionBack ? AliyunIRecorderTorchModeOn : AliyunIRecorderTorchModeOff;
[self.aliyunRecorder switchTorchWithMode:torchMode];
// 切换前后摄像头
[self.aliyunRecorder switchCameraPosition];
// 调整变焦倍数
self.aliyunRecorder.videoZoomFactor = 30.0;
// 调整摄像头角度
self.aliyunRecorder.cameraRotate = 270;
// 改变视频分辨率
[self.aliyunRecorder reStartPreviewWithVideoSize:CGSizeMake(720, 720)];
```

结束预览

通常在完成录制后，调用结束预览。

```
[self.aliyunRecorder stopPreview];
```

开始录制

startRecording和stopRecording需要成对出现，可以调用一次或多次，对应SDK内部会生成一段或多段临时视频文件。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 开始录制一段视频
[self.aliyunRecorder startRecording];
// 停止录制一段视频
[self.aliyunRecorder stopRecording];
```

结束录制

结束录制时可生成一个片段拼接的视频或只生成片段视频的配置信息。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

- finishRecording：拼接视频片段，生成完整视频。如果后续不需要继续编辑生成的视频，可以使用该方式。
- finishRecordingForEdit：不拼接视频片段。适用于拍摄视频后需要继续编辑视频，可以通过输出的taskPath初始化编辑器，详情请参考编辑初始化。

```
//结束录制，并且将录制片段视频拼接成一个完整的视频
[self.aliyunRecorder finishRecording];
//结束录制，不拼接视频片段，生成taskPath
NSString *taskPath = [self.aliyunRecorder finishRecordingForEdit];
AliyunEditor *editor = [[AliyunEditor alloc] initWithPath:taskPath preview:preview];
...
```

录制片段管理

配置录制最大或最小时长、删除录制片段、获取录制片段数量等，可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 删除最后一个视频片段
[self.aliyunRecorder.clipManager deletePart];
// 删除所有视频片段
[self.aliyunRecorder.clipManager deleteALLPart];
// 获取总的片段数量
[self.aliyunRecorder.clipManager partCount];
```

设置特效

配置美颜、滤镜、背景音乐等录制特效，可按需配置。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

滤镜

支持自定义滤镜，滤镜的制作方法请参见滤镜及转场。

```
//添加滤镜
NSString *filterDir = [self.class resourcePath:@"Filter/Jiaopian"];
AliyunEffectFilter *filter = [[AliyunEffectFilter alloc] initWithFile:filterDir];
[self.aliyunRecorder applyFilter:filter];
//移除滤镜
[self.aliyunRecorder deleteFilter];
```

动效滤镜

```
//添加动效滤镜
NSString *filterDir = [self.class resourcePath:@"AnimationEffect/split_screen_3"];
AliyunEffectFilter *animationFilter =[[AliyunEffectFilter alloc] initWithFile:filterDir];
[self.aliyunRecorder applyAnimationFilter:animationFilter];
//移除动效滤镜
[self.aliyunRecorder deleteAnimationFilter];
```

人脸贴纸

内置人脸检测添加人脸贴纸

```
// 使用内置人脸检测方法添加人脸贴纸
self.aliyunRecorder.useFaceDetect = YES;//开启人脸识别，当系统检测有人脸动图加入时将自动追踪
self.aliyunRecorder.faceDetectCount = 3;//设置人脸个数，最大设置3个，最小设置1个。如果不需要检测人脸，请使用:useFaceDetect = NO
self.aliyunRecorder.faceDectectSync = YES;//开启同步贴合人脸，同步贴合人脸的贴合性强，但是性能差的设备可能卡顿。非同步贴合人脸的画面流畅，但是贴合性差。
NSString *parsterDir = [self.class resourcePath:@"Gif/hanfumei-800"];
AliyunEffectPaster *paster = [[AliyunEffectPaster alloc] initWithFile:parsterDir];
[self.aliyunRecorder applyPaster:paster];//添加人脸贴纸
//删除人脸贴纸
[self.aliyunRecorder deletePaster:paster];
```

自定义人脸识别

```
// 第一步 关闭内置人脸识别
self.aliyunRecorder.useFaceDetect = NO;
// 第二步 在回调接口调用第三方人脸检测库进行人脸识别，并返回相关结果
- (void)recorderOutputVideoRawSampleBuffer:(CMSampleBufferRef)sampleBuffer {
    NSArray<AliyunFacePoint *> *facePoints = ...; // 第三方人脸检测库进行检测
    if (self.aliyunRecorder.faceNumbersCallback) {
        int num = (int)facePoints.count;
        self.aliyunRecorder.faceNumbersCallback(num);
    }
    [self.aliyunRecorder faceTrack:facePoints];
}
```

水印

设置水印的本质即加入一张图片进行渲染。可以通过applyImage接口添加静态贴纸进行渲染，例如实现相框的拍摄场景。

```
//添加水印
NSString *watermarkPath = [self.class resourcePath:@"Image/watermark.png"];
AliyunEffectImage *watermark = [[AliyunEffectImage alloc] initWithFile:watermarkPath];
CGSize size = CGSizeMake(280, 200);
CGFloat centerx = 44 + size.width / 2.0;
CGFloat centery = 44 + size.height / 2.0;
watermark.frame = CGRectMake(centerx, centery, size.width, size.height); // 注意frame的origin表示中点位置
[self.aliyunRecorder applyImage:watermark];
//删除水印
[self.aliyunRecorder deleteImage:watermark];
```

音乐

> **说明** 添加音乐后将无法设置静音，且不能进行录音。

```
//添加音乐
AliyunEffectMusic *effectMusic =[[AliyunEffectMusic alloc] initWithFile:[self.class resourcePath:@"bgm.aac"]];
effectMusic.startTime = 3.0; // 从音乐3s处开始播放
effectMusic.duration = 5.0;  // 持续5s，如果不设置则为音乐自身剩余长度且不能超过360s
[self.aliyunRecorder applyMusic:effectMusic];
//移除音乐
[self.aliyunRecorder applyMusic:nil];
```

高级美颜

视频录制模块，提供了基础的内置美颜功能，同时也支持使用外置的美颜SDK功能，如阿里云美颜特效SDK、相芯科技（FaceUnity）等美颜SDK。内置美颜功能相对比较简单，仅能设置不同的美颜等级；外置美颜SDK通常提供了更为丰富的美颜、美型、美妆美化、滤镜贴纸等功能。

- 内置美颜

```
// 开启
self.aliyunRecorder.beautifyStatus = YES;
self.aliyunRecorder.beautifyValue = 80;
// 关闭
self.aliyunRecorder.beautifyStatus = NO;
self.aliyunRecorder.beautifyValue = 0;
```

- 外置美颜SDK

  想要在短视频SDK中使用外置美颜SDK所提供的特效，则需要提前获取相应外置美颜SDK的权限并将外置美颜SDK集成接入到短视频SDK中。

  - 阿里云美颜特效SDK的集成接入等相关操作请参见美颜特效SDK。特效设置代码示例请参见使用示例。
  - FaceUnity的购买、集成、使用等相关操作，请参见FaceUnity。

  短视频SDK在摄像头采集后，会把采集的数据（CMSampleBufferRef）通过回调接口AliyunIRecorderDelegate传递给业务层，在业务层通过引入外置美颜SDK进行自定义渲染，最终把渲染的结果（CVPixelBufferRef）又交回给短视频SDK进行预览和合成。注意外置美颜SDK与短视频的GL环境冲突。

```
// AliyunIRecorderDelegate
- (CVPixelBufferRef)customRenderedPixelBufferWithRawSampleBuffer:(CMSampleBufferRef)sampl
eBuffer {
    if (!self.enableBeauty) {
        return CMSampleBufferGetImageBuffer(sampleBuffer);  // 不进行自定义渲染，直接返回
    }
    // 自定义渲染，这里可以接入美颜特效SDK，处理后返回CVPixelBufferRef示例
    ...
}
```

拍照

```
// 拍摄一张图片 异步获取
// image：采集的渲染后图片
// rawImage：采集的原始图片
[self.aliyunRecorder takePhoto:^(UIImage *image, UIImage *rawImage) {
}];
```

事件处理

对于常见的回调事件处理及对锁屏、来电、退后台等事件的特殊处理。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

回调事件处理

```
// 常见的事件回调处理如下
- (void)recorderDeviceAuthorization:(AliyunIRecorderDeviceAuthor)status {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (status == AliyunIRecorderDeviceAuthorAudioDenied) {
            [DeviceAuthorization openSetting:@"麦克风无权限"];
        } else if (status == AliyunIRecorderDeviceAuthorVideoDenied) {
            [DeviceAuthorization openSetting:@"摄像头无权限"];
        }
    });
}
- (void)recorderVideoDuration:(CGFloat)duration {
    // 这里更新录制进度
    NSLog(@"Record Video Duration: %f", duration);
}
- (void)recorderDidStopRecording {
    NSLog(@"Record Stop Recording");
    if (self.recordState == RecorderStateFinish) {
        if (self.aliyunRecorder.clipManager.duration >= self.aliyunRecorder.clipManager.min
Duration) {
            [self.aliyunRecorder finishRecording];
        }
        else {
            self.recordState = RecorderStatePreviewing;
        }
    }
}
- (void)recorderWillStopWithMaxDuration {
    NSLog(@"Record Will Stop Recording With Max Duration");
    [self.aliyunRecorder stopPreview];
}
- (void)recorderDidStopWithMaxDuration {
    NSLog(@"Record Did Recording With Max Duration");
    [self.aliyunRecorder finishRecording];
}
- (void)recorderDidFinishRecording {
    NSLog(@"Record Did Finish Recording");
    [self.aliyunRecorder stopPreview];
    self.recordState = RecorderStateInit;
}
- (void)recoderError:(NSError *)error {
    NSLog(@"Record Occurs Error: %@", error);
}
```

其他事件处理

对于锁屏、来电、退后台等事件，需要进行特殊处理。页面需要监听UIApplicationWillResignActiveNotification事件，当进入Inactive状态前需要调用stopRecording和stopPreview方法停止预览。还需要监听UIApplicationDidBecomeActiveNotification事件，当进入Active状态后调用startPreview方法重新开启预览。

5.4.3. 视频合拍

短视频SDK提供视频合拍功能，使用一个已有视频作为样本视频，与摄像头采集的数据按照特定的布局方式（例如左右分屏，上下分屏，画中画等）进行合拍录制，合拍视频的每一帧画面将会同时包含两路视频的画面，而音频则采用样本视频的音频。视频合拍是基础录制的功能升级，相比基础录制，视频合拍增加了一个新的本地视频轨道。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

概念介绍

在以下文档介绍中将提及一些特殊概念，为方便开发者理解，可预先对视频合拍、轨道及轨道布局的概念做相关了解。

相关类功能

名称	功能
AliyunMixRecorder	合拍功能核心类，包括录制、设置预览、设置特效、设置回调等视频合拍的核心录制功能。
AliyunMixMediaInfoParam	合拍参数配置类，包括摄像头窗口的位置大小，样本播放视频的位置大小等参数。
AliyunMixMediaRecordVideoInfo	采集视频配置类，设置视频采集分辨率。
AliyunMixMediaFileVideoInfo	样本视频配置类，设置样本视频的文件路径、参与合拍的视频开始时间和结束时间。

合拍流程

> **说明** 合拍录制功能需要获取摄像头和麦克风权限，否则无法录制。

视频合拍流程与基础录制流程基本相同，主要差别在于配置录制参数时设置输入输出参数以及设置预览View上的差别。

阶段	流程	说明	示例代码
基础	1	创建录制接口，并配置录制参数。	创建合拍参数配置
	2	回调设置。	回调设置
	3	设置开启或结束预览。	预览控制
	4	开始录制或停止录制。	开始录制

阶段	流程	说明	示例代码
	5	创建结束录制相关信息。	结束录制
进阶	6	视频合拍同基础录制一样，同样支持配置美颜、滤镜、背景音乐等录制特效，支持设置拍照等功能。	其他设置

创建合拍参数配置

设置合拍参数，作为合拍实例的初始化参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

设置合拍参数

```
AliyunMixMediaInfoParam *mixMediaInfo = [[AliyunMixMediaInfoParam alloc] init];
mixMediaInfo.outputSizeView = previewView;
mixMediaInfo.mixVideoInfo.filePath = mixVideoFile; // 样本视频路径
```

设置布局参数

合拍两路视频布局的坐标系与系统一致，此处以设置画中画布局为例。

```
// 设置样本视频窗口大小与位置：置于底层，覆盖全窗口
CGFloat mixWidth = previewView.bounds.size.width;
CGFloat mixHeight = previewView.bounds.size.height;
mixMediaInfo.mixVideoInfo.frame = CGRectMake(0, 0, mixWidth, mixHeight);
mixMediaInfo.mixVideoInfo.layerLevel = 1;
// 设置拍摄视频，比例为竖屏方向9：16，录制分辨率为360P，放在右下角，在样本视频上方
CGFloat recordRatio = 9.0 / 16.0;
mixMediaInfo.recordVideoInfo.resolution = CGSizeMake(360, 360 / recordRatio);
CGFloat recordHeight = previewView.bounds.size.height * 0.5;
CGFloat recordWidth = recordHeight * recordRatio;
mixMediaInfo.recordVideoInfo.frame = CGRectMake(previewView.bounds.size.width - recordWidth
- 4.0, previewView.bounds.size.height - recordHeight - 4, recordWidth, recordHeight);
mixMediaInfo.recordVideoInfo.layerLevel = 2;
```

设置其他参数

```
// 样本视频边框设置，非必须
mixMediaInfo.mixVideoInfo.borderInfo.color = UIColor.blueColor;
mixMediaInfo.mixVideoInfo.borderInfo.width = 2;
mixMediaInfo.mixVideoInfo.borderInfo.cornerRadius = 6.0;
// 拍摄视频边框设置，非必须
mixMediaInfo.recordVideoInfo.borderInfo.color = UIColor.blueColor;
mixMediaInfo.recordVideoInfo.borderInfo.width = 1;
mixMediaInfo.recordVideoInfo.borderInfo.cornerRadius = 6.0;
```

创建合拍对象

创建合拍对象时与基础录制的参数类似，更多信息请参见基础录制。

```
AliyunMixRecorder *_recorder = [[AliyunMixRecorder alloc] initWithMediaInfo:mixMediaInfo ou
tputSize:CGSizeMake(720, 720)];
_recorder.outputType = AliyunIRecorderVideoOutputPixelFormatType420f;
_recorder.useFaceDetect = YES;
_recorder.faceDetectCount = 2;
_recorder.faceDectectSync = NO;
_recorder.frontCaptureSessionPreset = AVCaptureSessionPreset1280x720;
_recorder.GOP = 250;
_recorder.videoQuality = AliyunVideoQualityHight;
_recorder.recordFps = 30;
_recorder.outputPath = [taskPath stringByAppendingPathComponent:@"output.mp4"];
_recorder.cameraRotate = 0;
_recorder.beautifyStatus = YES;
_recorder.frontCameraSupportVideoZoomFactor = YES;
self.aliyunMixRecorder = _recorder;
```

设置拍摄时长

> **说明** 最终合拍合成视频的时长，取决于两路视频的最长的时长。假设录制视频时长为10s，样本视频时长为5s，那么输出的合拍视频时长为10s。

```
// 建议合拍最长时长与样本视频时长一致，达到最大时长时，触发recorderDidStopWithMaxDuration回调
AVURLAsset *asset = [AVURLAsset assetWithURL:[NSURL fileURLWithPath:[self resourcePath:mixV
ideoPath]]];
[self.aliyunMixRecorder setRecordMaxDuration:asset.aliyunVideoDuration];
[self.aliyunMixRecorder setRecordMinDuration:1.0];
```

设置合拍背景

```
// 设置合拍颜色
[self.aliyunMixRecorder setBackgroundColor:0xFF0100];
// 设置合拍背景图片
NSString *imgPath = [self.class resourcePath:@"pig.jpeg"];
[self.aliyunMixRecorder setBackgroundImageFilePath:imgPath imageDisplayMode:AliyunMixVideoB
ackgroundImageModeScaleAspectFit];
```

设置输出音频

支持设置录音回声消除效果，支持两路音频混音、单路输出，或静音。

> **说明** 如果输出视频静音，或仅输出样本视频音频，可以不开启回声消除。

```
// 设置硬件回声消除效果，推荐使用Hardware模式
self.aliyunMixRecorder.recorderAECType = AliyunIRecorderAECTypeHardware;
// 设置合成视频使用录制音轨
[self.aliyunMixRecorder setMixAudioSource:MixAudioSourceTypeBoth];
[self.aliyunMixRecorder setMixAudioOriginalWeight:50 recordWeight:50];
```

回调设置

通过设置回调，及时获取音视频处理的进展和状态。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
- (void)recorderDeviceAuthorization:(AliyunIRecorderDeviceAuthor)status {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (status == AliyunIRecorderDeviceAuthorAudioDenied) {
            [DeviceAuthorization openSetting:@"麦克风无权限"];
        } else if (status == AliyunIRecorderDeviceAuthorVideoDenied) {
            [DeviceAuthorization openSetting:@"摄像头无权限"];
        }
    });
}
// 录制进度
- (void)recorderVideoDuration:(CGFloat)duration {
    NSLog(@"Mix Record Video Duration: %f", duration);
}
// 停止一段录制
- (void)recorderDidStopRecording {
    NSLog(@"Mix Record Stop Recording");
}
// 录制达到最大录制时长而停止，可以结束录制
- (void)recorderDidStopWithMaxDuration {
    NSLog(@"Mix Record Stop Recording With Max Duration");
    [self.aliyunMixRecorder finishRecording]; // 完成录制
}
// 录制完成
- (void)recorderDidFinishRecording {
    NSLog(@"Mix Record Did Finish Recording");
}
// 合拍完成合成
- (void)mixRecorderComposerDidComplete {
    NSLog(@"Mix Record Complete");
    [[MBProgressHUD HUDForView:self.view] hideAnimated:YES];
}
// 合成出现错误
- (void)mixRecorderComposerDidError:(int)errorCode {
    NSLog(@"Mix Record Error");
    [[MBProgressHUD HUDForView:self.view] hideAnimated:YES];
}
// 合拍开始进行合成
- (void)mixRecorderComposerDidStart {
    NSLog(@"Mix Record Start");
    MBProgressHUD *hud = [MBProgressHUD showHUDAddedTo:self.view animated:YES];
    hud.mode = MBProgressHUDModeDeterminate;
    hud.removeFromSuperViewOnHide = YES;
    hud.label.text = @"合成中...";
}
// 合拍合成结束
- (void)mixRecorderComposerOnProgress:(CGFloat)progress {
    NSLog(@"Mix Record Progress: %f", progress);
    MBProgressHUD *hub = [MBProgressHUD HUDForView:self.view];
    hub.progress = progress / self.aliyunMixRecorder.recordDuration;
}
```

预览控制

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 开启预览
[self.aliyunMixRecorder startPreviewWithPositon:AliyunIRecorderCameraPositionFront];
// 结束预览，通常在完成录制后，调用结束预览
[self.aliyunMixRecorder stopPreview];
```

开始录制

startRecording和stopRecording需要成对出现，可以调用一次或多次，对应SDK内部会生成一段或多段临时视频文件。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 开始录制一段视频
[self.aliyunMixRecorder startRecording];
// 停止录制一段视频
[self.aliyunMixRecorder stopRecording];
```

结束录制

录制完成后，调用finishRecording接口进行两路视频合成。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 结束录制，进行两路视频合成
[self.aliyunMixRecorder finishRecording];
```

其他设置

视频合拍同基础录制一样，同样支持配置美颜、滤镜、背景音乐等录制特效，支持设置拍照等功能，通过AliyunMixRecorder类来控制，其接口和基础录制的接口设置方法类似，更多信息请参见基础录制。

5.4.4. 多源录制

短视频SDK在基础录制功能的基础上全新升级了录制模块，除了满足基础录制所有的录制能力，还新增支持了View录制（即录屏）。同时也可以把View录制、摄像头录制、本地视频等作为输入源进行实时合成录制，满足各种实时录制合成的场景。例如，摄像头录制+View录制可以实现白板教学场景；摄像头录制+本地视频可以实现合拍场景。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

概念介绍

在以下文档介绍中将提及一些特殊概念，为方便开发者理解，请预先对多源录制、轨道及轨道布局的概念做相关了解。

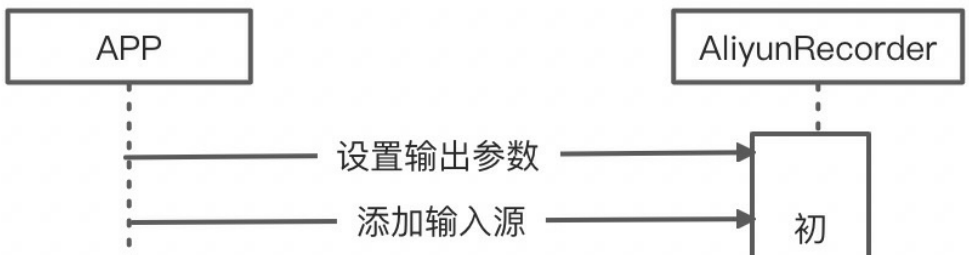
相关类功能

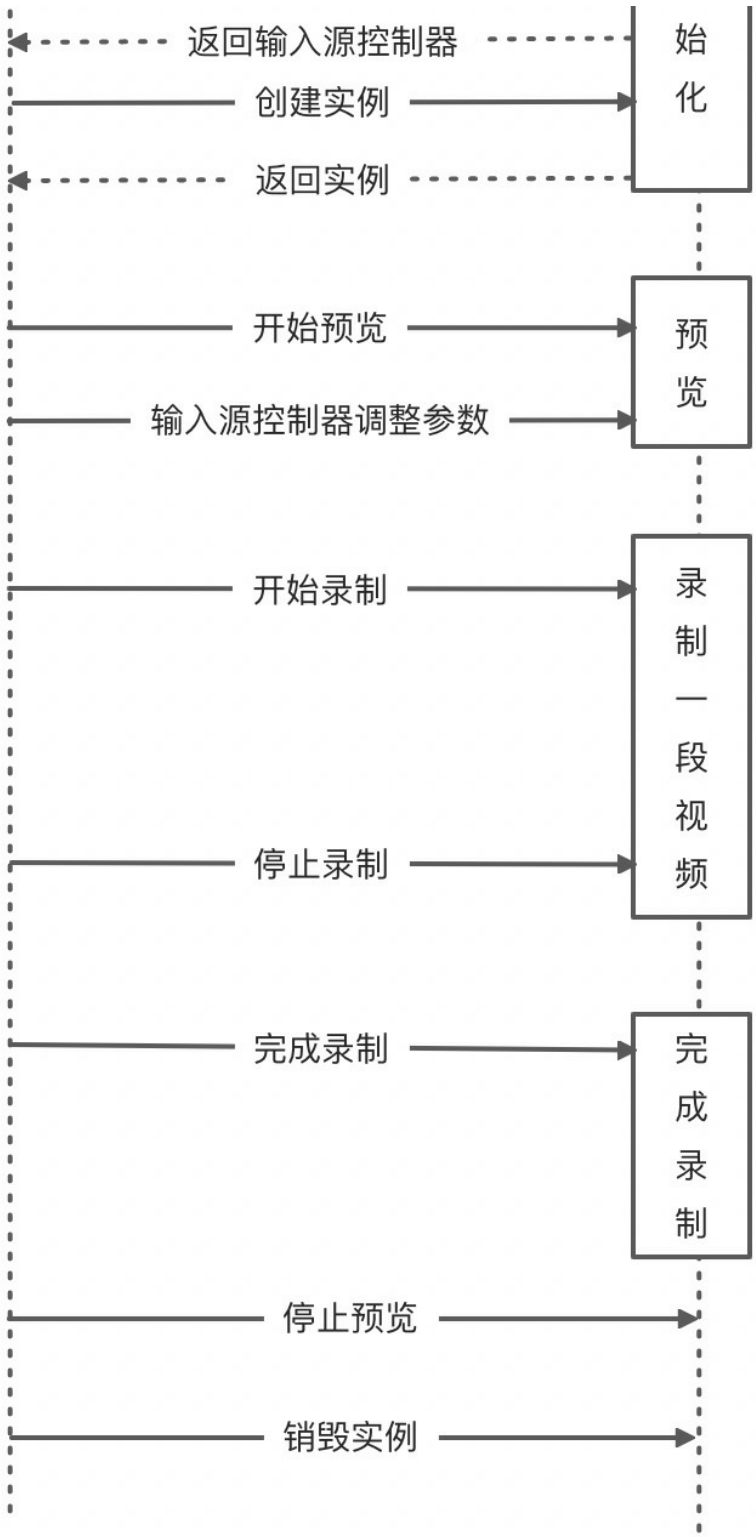
类名	功能
AliyunRecorder	多源录制核心类，包括录制、设置预览、设置特效、设置回调等多源录制的核心功能。
AliyunRecorderConfig	多源录制的配置类，设置录制的输出路径、添加采集源、添加水印、设置背景音乐等。
AliyunRecorderVideoConfig	输出视频配置类，输出视频的画面参数（分辨率、帧率、旋转角度、画面填充模式），压缩参数（编码、GOP、码率、视频质量，通常情况下不必设置，除非有相关要求）。
AliyunMicRecordController	麦克风控制器协议。
AliyunVideoRecordLayoutParam	输入源布局参数类，设置视频层级、自适应方式。
AliyunCameraRecordSource	摄像头录制输入源类。
AliyunCameraRecordController	摄像头录制控制器协议。
AliyunViewRecordSource	View录制输入源类。
AliyunViewRecordController	View录制控制器协议。
AliyunPlayerRecordSource	本地视频播放输入源配置类。
AliyunPlayerRecordController	本地视频播放控制器协议。
AliyunClipManager	录制片段管理器，获取片段信息，对视频片段进行删除操作等。
AliyunRecorderDelegate	录制代理回调。

多源录制流程

> **说明** 多源录制功能需要获取摄像头和麦克风权限，否则无法录制。

多源录制的流程如下图所示：





阶段	流程	说明	示例代码
	1	创建录制接口，并配置录制参数，添加录制源。	初始化

阶段	流程	说明	示例代码
基础	2	开启或结束预览。	预览控制
	3	开始或停止录制。	开始录制
	4	创建结束录制相关信息。	结束录制
进阶	5	配置输入源相关参数。	输入源控制-摄像头录制控制 输入源控制-View录制控制 输入源控制-本地视频控制
	6	设置背景音乐、背景图片、变速、自定义渲染等。	其他功能
	7	录制回调。	录制回调

初始化

初始化AliyunRecorder类，创建录制接口，并配置录制参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

设置输出参数

通过AliyunRecorderVideoConfig设置输出视频的画面参数（分辨率、帧率、旋转角度、画面填充模式），压缩参数（编码、GOP、码率、视频质量，一般情况下不必设置，除非有相关要求）。

```
AliyunRecorderVideoConfig *videoConfig = [[AliyunRecorderVideoConfig alloc] init];
videoConfig.resolution = CGSizeMake(720, 1280);           // 720P
videoConfig.fps = 30;
...
```

添加输入源

先通过输出参数对象与outputPath创建配置对象，然后添加输入源。

> **说明**
>
> - 可根据业务场景组合多个采集源，可以添加一个或多个，但不建议超过3个。
> - 只能添加一个摄像头录制源，重复添加也无效。

```
//创建配置对象
AliyunRecorderConfig *config = [[AliyunRecorderConfig alloc] initWithVideoConfig:videoConfi
g outputPath:[taskPath stringByAppendingPathComponent:@"output.mp4"]];
//添加摄像头录制源
// 指定摄像头布局参数
AliyunVideoRecordLayoutParam *cameraLayout = [[AliyunVideoRecordLayoutParam alloc] initWith
RenderMode:AliyunRenderMode_ResizeAspectFill];
cameraLayout.size = resolution;
cameraLayout.center = CGPointMake(resolution.width / 2.0, resolution.height / 2.0);
cameraLayout.zPosition = 1;
// 添加录制源，返回控制器
self.cameraRecorderController = [config addCamera:cameraLayout];
self.cameraRecorderController.preview = self.videoView;
self.cameraRecorderController.camera.resolution = AliyunRecordCameraResolution_3840x2160;
self.cameraRecorderController.camera.position = AVCapturePositionBack;
//添加View录制源
// 指定View录制布局参数
AliyunVideoRecordLayoutParam *viewRecordLayout = [[AliyunVideoRecordLayoutParam alloc] init
WithRenderMode:AliyunRenderMode_ResizeAspect];
viewRecordLayout.size = CGSizeMake(0.5*videoConfig.resolution.width, 0.5*videoConfig.resolu
tion.height);
viewRecordLayout.center = CGPointMake(0.5*videoConfig.resolution.width, 0.5*videoConfig.res
olution.height);
viewRecordLayout.zPosition = 2;
// 指定View录制源参数
AliyunViewRecordSource *viewSource = [[AliyunViewRecordSource alloc] initWithTargetView:dra
wView fps:videoConfig.fps];
viewSource.captureInBackground = YES;
// 添加录制源，返回控制器
self.viewRecordController = [config addViewSource:viewSource layout:viewRecordLayout];
//添加本地播放源
// 指定View录制布局参数
AliyunVideoRecordLayoutParam *playerRecordLayout = [[AliyunVideoRecordLayoutParam alloc] in
itWithRenderMode:AliyunRenderMode_ResizeAspect];
playerRecordLayout.size = CGSizeMake(0.5*videoConfig.resolution.width, 0.5*videoConfig.reso
lution.height);
playerRecordLayout.center = CGPointMake(0.5*videoConfig.resolution.width, 0.5*videoConfig.r
esolution.height);
playerRecordLayout.zPosition = 2;
// 指定View录制源参数
AVURLAsset *asset = [AVURLAsset URLAssetWithURL:[NSURL fileURLWithPath:videoPath] options:n
il];
AliyunPlayerRecordSource *playerSource = [[AliyunPlayerRecordSource alloc] initWithAsset:as
set fps:videoConfig.fps];
// 添加录制源，返回控制器
self.playerRecordController = [config addMVSource:playerSource layout:playerRecordLayout];
self.playerRecordController.preview = self.playerView;
```

创建实例

```
// 通过输入配置创建
AliyunRecorder *recorder = [[AliyunRecorder alloc] initWithConfig:config];
recorder.delegate = self;
recorder.clipManager.maxDuration = 5;
recorder.clipManager.minDuration = 1;
recorder.clipManager.deleteVideoClipsOnExit = YES;
self.aliyunRecorder = recorder;
```

预览控制

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//开启预览
[self.aliyunRecorder startPreview];
//结束预览，通常在完成录制后，调用结束预览
[self.aliyunRecorder stopPreview];
```

开始录制

startRecording和stopRecording需要成对出现，可以调用一次或多次，对应SDK内部会生成一段或多段临时视频文件。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 开始录制一段视频
[self.aliyunRecorder startRecord];
// 停止录制一段视频，并非立即停止状态，会出现录制状态的切换，注意通过onAliyunRecorder:stateDidChange:获取录制状态的回调
[self.aliyunRecorder stopRecord];
```

录制状态及说明如下：

状态	说明
AliyunRecorderState_Idle	空闲状态，等待录制。
AliyunRecorderState_LoadingForRecord	录制前加载，等待各个录制源首帧回调；收到全部回调后会自动变为Recording状态。
AliyunRecorderState_Recording	录制中。
AliyunRecorderState_Stopping	正在停止，等待内部缓冲处理完成会自动变为停止状态。
AliyunRecorderState_Stop	停止录制。
AliyunRecorderState_Error	发生错误，可以通过cancel来把状态重置为Idle。

结束录制

结束录制时可生成一个片段拼接的视频或只生成片段视频的配置信息。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

- finishRecord：拼接视频片段，生成完整视频。如果后续不需要继续编辑生成的视频，可以使用该方式。
- finishRecordForEdit：不拼接视频片段。适用于拍摄视频后需要继续编辑视频，可以通过输出的taskPath

初始化编辑器，详情请参考编辑初始化。

```
// 结束录制，并且将录制片段视频拼接成一个完整的视频
[self.aliyunRecorder finishRecord:^(NSString *outputPath, NSError *error) {
    if (!error) {
        // 录制完成，可以把outputPath的视频进行预览、编辑、上传等
    }
}];
// 结束录制，不拼接视频片段，生成taskPath
[self.aliyunRecorder finishRecordForEdit:^(NSString *taskPath, NSError *error) {
    if (!error) {
        // 录制完成，后续通过taskPath初始化编辑器(AliyunEditor)进行视频编辑，详情参考编辑模块
    }
}];
```

输入源控制-摄像头录制控制

通过添加源返回基于AliyunCameraRecordController协议的控制器，可以执行参数调整、边框调整、基本美颜、人脸贴纸、静态贴纸、动图、静态滤镜、动效滤镜、拍照等操作。

常用参数调整

```
// 手电筒
self.cameraRecorderController.camera.torchMode = AVCaptureTorchModeOn;
// 摄像头位置
self.cameraRecorderController.camera.position = AVCaptureDevicePositionBack;
// 变焦倍数
self.cameraRecorderController.camera.videoZoomFactor = 2.0;
// 曝光度
self.cameraRecorderController.camera.exposureValue = 0.8;
// 方向
self.cameraRecorderController.camera.orientation = UIDeviceOrientationLandscapeLeft;
// 采集分辨率，一般情况下不建议修改，会根据输出大小自动适配合适采集分辨率
self.cameraRecorderController.camera.resolution = AliyunRecordCameraResolution_1280x720;
// 闪光灯，拍照时启用
self.cameraRecorderController.camera.flashMode = AVCaptureFlashModeOn;
```

边框调整

```
AliyunVideoRecordBorderInfo *cameraBorder = [AliyunVideoRecordBorderInfo new];
cameraBorder.color = UIColor.whiteColor;
cameraBorder.width = 3.0;
cameraBorder.cornerRadius = 10.0;
self.cameraRecorderController.borderInfo = cameraBorder;
```

基本美颜

```
// 开启
self.cameraRecorderController.beautifyStatus = YES;
self.cameraRecorderController.beautifyValue = 80;
// 关闭
self.cameraRecorderController.beautifyStatus = NO;
self.cameraRecorderController.beautifyValue = 0;
```

人脸贴纸

```
//添加人脸贴纸
[self.cameraRecorderController applyFaceSticker:[self.class resourcePath:@"Gif/hanfumei-800
"]];
```

滤镜

支持自定义滤镜，滤镜的制作方法请参见滤镜及转场。

```
//添加滤镜
AliyunEffectFilter *filter = [[AliyunEffectFilter alloc] initWithFile:path];
[self.cameraRecorderController applyFilter:filter];
```

动效滤镜

```
//添加动效滤镜
NSString *filterDir = [self.class resourcePath:@"AnimationEffect/split_screen_3"];
AliyunEffectFilter *animationFilter =[[AliyunEffectFilter alloc] initWithFile:filterDir];
[self.cameraRecorderController applyAnimationFilter:animationFilter];
```

静态贴纸

```
//添加静态贴纸
AliyunImageStickerController *imageController = [self.cameraRecorderController addImageStic
ker:imagePath];
[imageController beginEdit];
imageController.image.center = CGPointMake(150, 200);
[imageController endEdit];
```

动态贴纸

支持自定义动态贴纸，动态贴纸的制作方法请参见动图。

```
//添加动态贴纸
AliyunGifStickerController * gifController = [self.cameraController addGifStickerWithConfig
:[PathTool boundlePathWithPath:@"Resource/Gif/hanfumei-800"]];
[gifController beginEdit];
gifController.gif.center = CGPointMake(150, 200);
[gifController endEdit];
```

拍照

```
 // 拍摄一张图片 异步获取
 // image 采集的渲染后图片
 // rawImage 采集的原始图片
[self.cameraRecorderController takePhoto:^(UIImage *image, UIImage *rawImage) {
}];
```

输入源控制-View录制控制

通过添加源返回基于AliyunViewRecordController协议的控制器，仅可以进行边框调整。

边框

```
AliyunVideoRecordBorderInfo *border = [AliyunVideoRecordBorderInfo new];
border.color = UIColor.whiteColor;
border.width = 3.0;
border.cornerRadius = 10.0;
self.viewRecorderController.borderInfo = border;
```

输入源控制-本地视频控制

通过添加源返回基于AliyunPlayerRecordController协议的控制器，仅可以进行边框调整。

边框

```
AliyunVideoRecordBorderInfo *border = [AliyunVideoRecordBorderInfo new];
border.color = UIColor.whiteColor;
border.width = 3.0;
border.cornerRadius = 10.0;
self.playerRecordController.borderInfo = border;
```

其他功能

支持添加背景音乐、添加水印、设置背景等功能。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

片段管理

通过 `startRecord` 和 `stopRecord` 来录制多个片段，片段可通过clipManager管理，支持回删最后一个片段或删除所有片段等操作。接口参数请参考AliyunClipManager。

```
// 删除最后一个视频片段
[self.aliyunRecorder.clipManager deletePart];
// 删除所有视频片段
[self.aliyunRecorder.clipManager deleteALLPart];
// 获取总的片段数量
[self.aliyunRecorder.clipManager partCount];
```

变速录制

```
//设置录制速度，建议为0.5到2之间。
[self.aliyunRecorder setRate:2];
```

背景音乐

```
// 添加背景音乐
AVURLAsset *audioAsset = [AVURLAsset URLAssetWithURL:[NSURL fileURLWithPath:filePath] optio
ns:nil];
float audioDuration = CMTimeGetSeconds(audioAsset.duration);
[self.aliyunRecorder.config setBgMusicWithFile:filePath
                                 startTime:0.0    // 从0秒开始
                                 duration:MIN(self.aliyunRecorder.clipManager.maxDuration, a
udioDuration)];
// 移除背景音乐
[self.aliyunRecorder.config removeBgMusic];
```

水印

```
// 生成水印
- (AliyunRecorderImageSticker *) waterMark
{
    if (!_waterMark) {
        NSString *watermarkPath = [AlivcImage pathOfImageName:@"shortVideo_paster_gif"];
        _waterMark = [[AliyunRecorderImageSticker alloc] initWithImagePath:watermarkPath];
        _waterMark.size = CGSizeMake(42, 30);
        _waterMark.center = CGPointMake(_waterMark.size.width * 0.5 + 4, _waterMark.size.he
ight * 0.5 + 4);
        _waterMark.autoresizingMask = UIViewAutoresizingFlexibleRightMargin | UIViewAutores
izingFlexibleBottomMargin;
    }
    return _waterMark;
}
// 应用水印
[self.aliyunRecorder.config addWaterMark:self.waterMark];
// 删除水印
[self.aliyunRecorder.config removeWaterMark:self.waterMark.stickerId];
```

设置背景

```
// 生成背景对象
- (AliyunRecorderBackgroundInfo *) bgInfo
{
    if (!_bgInfo) {
        _bgInfo = [AliyunRecorderBackgroundInfo new];
         // _bgInfo.color = UIColor.redColor;
        NSString *imgName = @"xxx.png";
        _bgInfo.image = [UIImage imageNamed:imgName];
        _bgInfo.renderMode = AliyunRenderMode_ResizeAspectFill;
    }
    return _bgInfo;
}
// 应用背景对象
self.aliyunRecorder.config.bgInfo = self.bgInfo;
```

自定义渲染

添加摄像头录制源后，会把采集的数据（CMSampleBufferRef）通过回调接口customRender传递给业务层，可以在业务层通过引入第三方SDK进行自定义渲染，最终把渲染的结果（CVPixelBufferRef）又交回给短视频SDK进行预览和合成。

```
// 第一步：设置自定义渲染，注意一定要有添加摄像头录制源
self.aliyunRecorder.customRender = self;
// 第二步：AliyunRecorderCustomRender协议
- (CVPixelBufferRef) onAliyunRecorderCustomRenderToPixelBuffer:(AliyunRecorder *)recorder w
ithSampleBuffer:(CMSampleBufferRef)sampleBuffer {
    // 自定义渲染，此处可以接入美颜特效SDK，处理后返回CVPixelBufferRef示例
    // 不进行自定义渲染，可以返回CMSampleBufferGetImageBuffer(sampleBuffer)
    ...
}
```

说明 自定义渲染可以把摄像头采集回来的数据进行任何加工，应用场景包括美颜、美型、美妆、手势识别、AI抠像、绿幕抠图，美颜特效SDK已提供了这些能力，并能与短视频SDK很好的结合使用，详细使用请参见iOS短视频SDK Demo。

录制回调

通过设置回调，及时获取音视频处理的进展和状态。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//常见的事件回调处理如下
#pragma mark - AliyunRecorderDelegate
- (void)onAliyunRecorderWillStopWithMaxDuration:(AliyunRecorder *)recorder {
    NSLog(@"Record2 Will Stop Recording With Max Duration");
    [self.aliyunRecorder stopPreview];
}
- (void)onAliyunRecorderDidStopWithMaxDuration:(AliyunRecorder *)recorder {
    NSLog(@"Record2 Did Recording With Max Duration");
    [self.aliyunRecorder finishRecord:^(NSString *outputPath, NSError *error) {
        if (!error) {
        }
    }];
}
- (void)onAliyunRecorder:(AliyunRecorder *)recorder progressWithDuration:(CGFloat)duration
{
    NSLog(@"Record2 Video Duration: %f", duration);
}
- (void)onAliyunRecorder:(AliyunRecorder *)recorder occursError:(NSError *)error {
    NSLog(@"Record2 Occurs Error: %@", error);
}
```

5.5. 视频裁剪

短视频SDK提供了裁剪模块，支持对视频按时长、画幅裁剪，对音频按时长裁剪，对图片按画幅裁剪。本文为您介绍iOS端短视频SDK的视频裁剪方法。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

类名	功能
AliyunCrop	视频音频裁剪功能核心类，包括对视频和音频的裁剪、设置裁剪参数、设置回调等裁剪核心功能。
AliyunImageCrop	图片裁剪功能核心类，包括对图片的裁剪、设置裁剪参数等裁剪核心功能。

注意 视频音频的裁剪是异步操作，AliyunCrop实例对象必须是成员变量，不能是局部变量。裁剪过程退到后台会裁剪失败。

视频裁剪-常规裁剪

常规裁剪会重新编解码，支持设置更多输出视频的参数，如输出分辨率、码率、帧率、关键帧间隔、编码方式、视频质量等。

接口参数请参考AliyunCrop。

初始化

```
//调用初始化方法创建裁剪对象
self.crop = [[AliyunCrop alloc] initWithDelegate:self];
```

设置输入路径，输出路径，裁剪时间

```
//设置视频的输入和输出文件路径
self.crop.inputPath = [self.class resourcePath:@"input.mp4"];
self.crop.outputPath =  [self.class resourcePath:@"output.mp4"];
//设置裁剪的时间起点和时间终点，单位：秒
self.crop.startTime = 0.0;
self.crop.endTime = 5.0;
```

说明 如果输出文件路径是多级目录，请确保目录已经创建。

裁剪参数配置

```
//输出视频分辨率
self.crop.outputSize = CGSizeMake(720, 720);
//裁剪区域
self.crop.rect = CGRectMake(0, (1280-720) / 2, 720, 720);
//裁剪模式
self.crop.cropMode = AliyunCropModeScaleAspectCut;
//码率bps
self.crop.bitrate = 1000 * 1000;
//帧率
self.cropl.fps = 30;
//关键帧间隔
self.crop.gop = 90;
//视频质量
self.crop.videoQuality = AliyunVideoQualityHight;
//设置编码模式为硬编
self.crop.encodeMode = 1;
```

开始裁剪

```
[self.crop startCrop];
```

裁剪回调

```
//裁剪出错
- (void)cropOnError:(int)error {
}
 //裁剪进度
- (void)cropTaskOnProgress:(float)progress {
}
//裁剪完成
- (void)cropTaskOnComplete {
}
//裁剪取消
- (void)cropTaskOnCancel {
}
```

视频裁剪-快速裁剪

快速裁剪不重新编解码，只对时长做简单处理，不支持设置其他视频参数。

接口参数请参考AliyunCrop。

初始化

```
//调用初始化方法创建裁剪对象
self.crop = [[AliyunCrop alloc] initWithDelegate:self];
```

设置为快速裁剪模式

```
//开启裁剪优化，设置为快速裁剪
self.crop.shouldOptimize = YES;
```

设置输入路径，输出路径，裁剪时间

```
//设置视频的输入和输出文件路径
self.crop.inputPath = [self.class resourcePath:@"input.mp4"];
self.crop.outputPath =  [self.class resourcePath:@"output.mp4"];
//设置裁剪的开始和结束时间
self.crop.startTime = 0.0;
self.crop.endTime = 5.0;
```

> **说明** 如果输出文件路径是多级目录，请确保目录已经创建。

开始裁剪

```
[self.crop startCrop];
```

裁剪回调

```
//裁剪出错
- (void)cropOnError:(int)error {
}
 //裁剪进度
- (void)cropTaskOnProgress:(float)progress {
}
//裁剪完成
- (void)cropTaskOnComplete {
}
//裁剪取消
- (void)cropTaskOnCancel {
}
```

音频裁剪

接口参数请参考AliyunCrop。

初始化

```
//调用初始化方法创建裁剪对象
self.crop = [[AliyunCrop alloc] initWithDelegate:self];
```

设置输入路径，输出路径，裁剪时间

```
//设置视频的输入和输出文件路径
self.crop.inputPath = [self.class resourcePath:@"input.mp3"];
self.crop.outputPath =  [self.class resourcePath:@"output.mp3"];
//设置裁剪的时间起点和时间终点，单位：秒
self.crop.startTime = 0.0;
self.crop.endTime = 5.0;
```

> **说明** 如果输出文件路径是多级目录，请确保目录已经创建。

开始裁剪

```
[self.crop startCrop];
```

裁剪回调

```
//裁剪出错
- (void)cropOnError:(int)error {
}
 //裁剪进度
- (void)cropTaskOnProgress:(float)progress {
}
//裁剪完成
- (void)cropTaskOnComplete {
}
//裁剪取消
- (void)cropTaskOnCancel {
}
```

图片裁剪

接口参数请参考AliyunImageCrop。

初始化

```
//调用初始化方法创建裁剪对象
AliyunImageCrop *imageCrop  = [[AliyunImageCrop alloc] init];
```

设置输入图片

```
imageCrop.originImage = image;
```

裁剪参数设置

```
//设置输出图片分辨率
imageCrop.outputSize = CGSizeMake(200, 200);
//（可选）设置裁剪区域（像素）
imageCrop.cropRect = CGRectMake(50, 0, 200, 200);
//（可选）设置图片裁剪模式
imageCrop.cropMode = AliyunImageCropModeAspectCut;
```

开始裁剪，生成裁剪图片

```
UIImage *outputImage = [imageCrop generateImage];
```

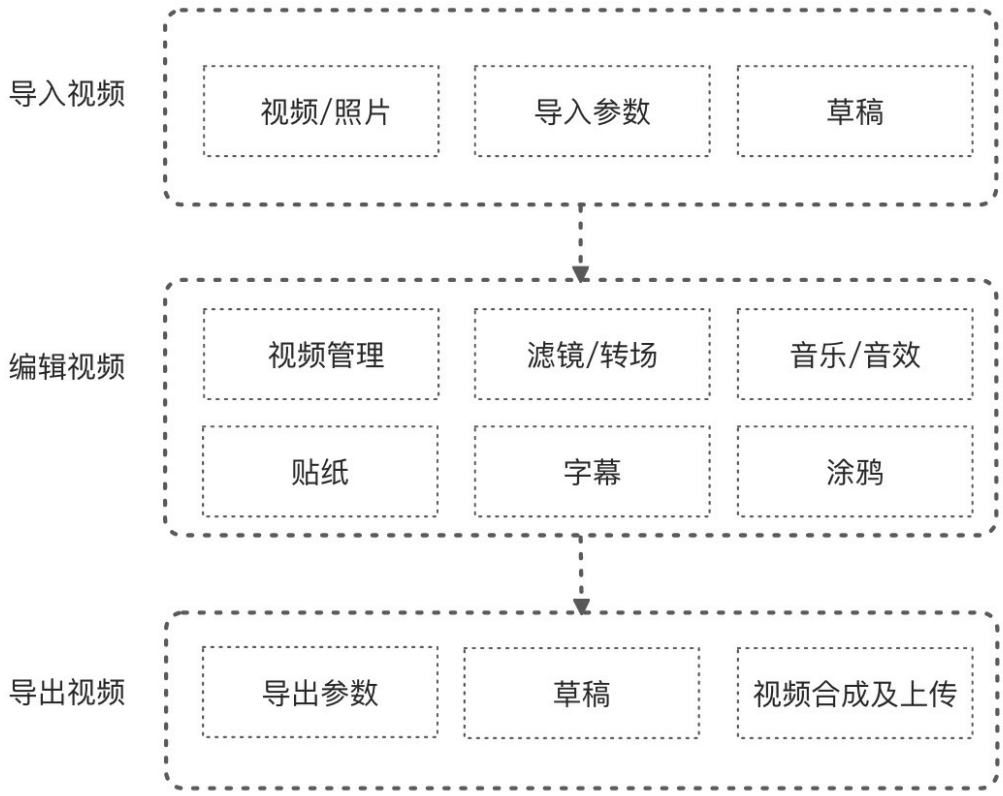
5.6. 视频编辑

5.6.1. 概述

短视频SDK提供视频编辑与导出功能，支持视频图片素材混合导入，提供滤镜、配音、时间特效、过渡效果等丰富的编辑效果。本文介绍视频编辑的流程及版本差异。

总体流程

编辑制作一个视频通常需要先导入视频，再编辑视频，最后导出视频。总体的流程及各环节涉及的功能点如下图所示：



版本支持

各版本支持的功能如下表所示，表格中的✓表示支持，×表示不支持。

功能	基础版	标准版	专业版
导入视频	×	✓	✓
导出视频	×	✓	✓
视频管理（编辑视频）	×	✓	✓
音乐及音效（编辑视频）	×	✓	✓
滤镜及转场（编辑视频）	×	✓	✓
滤镜及转场（编辑视频）	×	✓	✓

功能	基础版	标准版	专业版
时间特效（编辑视频）	×	✓	✓
画中画（编辑视频）	×	✓	✓
字幕（编辑视频）	×	×	✓
动态贴纸（编辑视频）	×	×	✓
MV（编辑视频）	×	×	✓

5.6.2. 导入视频

短视频SDK支持视频图片素材混合导入，支持本地媒体资源导入和草稿箱导入这两种方式。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

相关类功能

类名	功能
AliyunImporter	视频导入核心类，用于构建编辑初始化配置文件，配置媒体片段等。
AliyunIClipConstructor	媒体片段构造器协议工厂类，用于获取、增加、删除媒体片段等。
AliyunClip	媒体片段，用于设置媒体片段的开始结束时间、显示模式、旋转角度等信息。
AliyunVideoParam	视频参数设置类，设置视频的质量、填充模式、编码类型等参数。

本地媒体资源导入

本地媒体资源导入，即通过AliyunImporter类添加不同的媒体片段，最终生成初始任务配置的视频源文件，作为AliyunEditor类的输入参数。

```
//0. 指定配置文件夹路径和输出视频分辨率
NSString *taskPath = @"xxx";
//1. 创建实例
AliyunImporter *importer = [[AliyunImporter alloc] initWithPath:taskPath outputSize:outputR
esolution];
//2.1 添加视频
AliyunClip *videoClip = [[AliyunClip alloc] initWithVideoPath:@"your video path" animDurati
on:0];
[importer addMediaClip:videoClip];
//3. 设置输出参数
AliyunVideoParam *param = [[AliyunVideoParam alloc] init];
param.fps = 30; // 帧率
param.gop = 90; // 关键帧间隔
param.videoQuality = AliyunVideoQualityHight; // 视频质量
param.scaleMode = AliyunScaleModeFill; // 缩放模式
param.codecType = AliyunVideoCodecHardware; // 编码模式
[importer setVideoParam:param];
//4. 生成视频源配置文件
[importer generateProjectConfigure];
//5.创建editor
 self.editor = [[AliyunEditor alloc] initWithPath:taskPath
                                          preview:self.preView];
```

草稿箱导入

编辑状态最终会以工程配置的形式记录下来保存到草稿箱中，只要提供工程配置文件就能从草稿箱中还原编辑状态，详细操作请参见草稿箱。

5.6.3. 编辑视频

短视频SDK提供视频编辑功能，支持视频图片素材混合导入、滤镜、配音、时间特效、画中画等丰富的编辑效果。本文介绍iOS端短视频SDK视频编辑的流程及方法。

版本支持

版本	是否支持
专业版	支持所有功能。
标准版	部分支持，支持除字幕、动态贴纸、MV外的其他功能。
基础版	不支持。

相关类功能

操作	类名	功能
初始化	AliyunEditor	视频编辑核心类。
	AliyunIClipConstructor	视频源管理器。
	AliyunClip	媒体片段。

操作	类名	功能
开始编辑		
	AEPVideoTrack	工程配置里的主流轨道。
	AEPVideoTrackClip	工程配置里的主流片段。
预览控制	AliyunIPlayer	播放协议。
	AliyunIPlayerCallback	播放状态回调协议。
设置视频特效	AliyunEffectFilter	滤镜效果model类。
	AliyunEffectTimeFilter	时间特效model类。
	AliyunTransitionEffect	转场效果的基类。
	AliyunEffectMV	MV效果model类。
	AliyunFilterManager	滤镜管理器，可管理lut滤镜和静态滤镜。
	AliyunLutFilterController	lut滤镜控制器。
	AliyunLutFilter	lut滤镜Model。
	AliyunShaderFilterController	静态滤镜控制器。
	AliyunShaderFilter	静态滤镜Model。
设置音乐及音效	AliyunEffectMusic	音乐。
	AliyunEffectDub	配音。
	AEPAudioTrack	工程配置里的音轨。
	AEPAudioTrackClip	工程配置里的音轨片段。
设置画中画	AliyunPipManager	画中画管理类。
	AliyunPipTrackController	画中画轨道控制器。
	AliyunPipClipController	画中画片段控制器。
	AliyunPipClip	画中画数据模型。
	AEPPipVideoTrackClip	工程配置中的画中画片段。
	AEPPipVideoTrack	工程配置中的画中画轨道。
	AliyunClipAugmentationInfo	画面增强信息。
	AliyunClipAudioInfo	音频相关信息。

操作	类名	功能
	AliyunPureColorBorderInfo	边框相关信息。
设置字幕及贴纸	AliyunStickerManager	贴纸及字幕管理器。
	AliyunCaptionStickerController	字幕控制器。
	AliyunGifStickerController	动图控制器。
	AliyunImageStickerController	静图控制器。
	AliyunCaptionSticker	字幕数据模型。
	AliyunGifSticker	动图数据模型。
	AliyunImageSticker	静图数据模型。
	AEPGifStickerTrack	工程配置里的动图轨道。
	AEPImageStickerTrack	工程配置里的静图轨道。
	AEPCaptionTrack	工程配置里的字幕轨道。
草稿箱	AliyunEditorProject	工程配置。
	AliyunDraft	草稿对象。
	AliyunDraftManager	本地草稿管理器。
	AliyunDraftLoadTask	草稿资源加载任务。
	AliyunDraftProjectUploadTask	草稿上传任务。
	AEPSource	资源对象。
	AEPResourceModel	加载任务中更具体的资源模型。
其他设置	AliyunICanvasView	涂鸦画布视图。
	AliyunIPaint	画笔。

编辑视频流程

阶段	流程	说明	示例代码
基础	1	创建并初始化编辑器。	初始化
	2	在编辑过程中动态裁剪视频、动态更换视频源、动态调整视频转场时间及转场效果。	视频管理

阶段	流程	说明	示例代码
	3	设置编辑器的预览播放。	预览控制
进阶	4	设置滤镜、转场及MV特效。	设置视频特效
	5	设置背景音乐、配音及音效。	设置音乐及音效
	6	设置画中画。	设置画中画
	7	设置字幕、花字、文字气泡及贴纸。	设置字幕及贴纸
	8	支持编辑草稿箱中的视频、或将编辑完成的视频保存至草稿箱。	草稿箱
	9	设置涂鸦。	其他设置

初始化

创建并初始化编辑器。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 1.任务路径：每一个媒体编辑状态会生成一个任务，编辑初始化时需要提供任务存储的路径，任务必须先初始化
NSString *taskPath = @"xxx"; // taskPath为导入视频的地址，通常为本地媒体资源导入或草稿箱导入
// 2.预览视图：设置预览视图后，编辑过程中，每一个操作都会实时地展示在一个预览视图上
UIView *preview = xxx;
// 3.实例化
AliyunEditor *editor = [[AliyunEditor alloc] initWithPath:taskPath preview:preview];
```

> **说明** 为了保证预览效果与输出效果保持一致，建议您的预览视图大小与最终输出视频的分辨率保持一样的宽高比。

视频管理

视频编辑器里的视频和图片最终会由视频源管理器AliyunIClipConstructor统一管理，通过AliyunIClipConstructor修改编辑器里的视频和图片后，需要下一次调用 `[editor startEdit]` 编辑后才能生效。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 1. 获取视频源管理器
id<AliyunIClipConstructor> contructor = [editor getClipConstructor];
// 2. 添加片段
// 2.1 先停止编辑
[editor stopEdit];
// 2.2 创建视频
AliyunClip *clip = [[AliyunClip alloc] initWithVideoPath:videoPath startTime:2 duration:6 a
nimDuration:0];
// 2.3.1 把视频追加到最后
[contructor addMediaClip:clip];
// 2.3.2 或者把视频加到指定位置
[contructor addMediaClip:clip atIndex:1];
// 2.4 开始编辑
[editor startEdit];
// 3. 更新片段（替换片段）
[contructor updateMediaClip:clip atIndex:1];
// 4. 删除片段
// 4.1 删除最后一个片段
[contructor deleteLastMediaClip];
// 4.2 删除指定的一个片段
[contructor deleteMediaClipAtIndex:1];
// 4.3 删除所有片段
[contructor deleteAllMediaClips];
// 5. 获取当前所有片段
NSArray<AliyunClip *> *mediaClips = contructor.mediaClips;
```

预览控制

在视频编辑过程中，提供一系列对当前视频的播放控制操作，如播放、暂停、获取当前时长等。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

播放控制

```
// 获取预览播放器
id<AliyunIPlayer> player = [editor getPlayer];
//开始播放
[player play];
//继续播放
[player resume];
// 暂停播放
[player pause];
// 跳转到指定位置
[player seek:timeInSecond];
// 静音
[editor setMute:YES];
// 设置音量
[editor setVolume:50];
//获取当前流的时间 - 不受时间特效影响
double currentTime = [player getCurrentStreamTime];
// 获取当前播放的时间 - 受时间特效影响
double currentTime = [player getCurrentTime];
// 获取流时长 - 不受时间特效影响
double duration = [player getStreamDuration];
// 获取播放总时长 - 受时间特效影响
double duration = [player getDuration];
```

播放回调

```
// 监听播放状态
editor.playerCallback = self;
// 协议
- (void)playerDidEnd {
    // 播放结束回调
}
- (void)playProgress:(double)playSec streamProgress:(double)streamSec {
    // 播放进度回调
}
- (void)playError:(int)errorCode {
    // 播放错误回调
}
```

设置视频特效

支持设置的视频特效包括滤镜、转场、MV及时间特效。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

滤镜

- 滤镜分为lut滤镜、静态滤镜和动效滤镜。
  - lut滤镜：使用Lookup Table方式进行像素替换。
  - 静态滤镜：通过编写着色语言方式进行像素计算，不支持带动画效果。
  - 动效滤镜：通过编写着色语言方式进行像素计算，带动画效果。
- 滤镜支持自定义制作，制作方法请参见滤镜及转场。

lut滤镜

```
// 1 添加lut滤镜
AliyunLutFilterController *controller = [[editor getFilterManager] applyLutFilterWithPath:p
ath intensity:intensity];
if (!controller) {
    // 添加LutFilter失败
}
// 2 更新intensity
controller.model.intensity = 0.5;
// 3 删除lut滤镜
[[editor getFilterManager] removeFilter:controller];
```

静态滤镜

```
// 1 添加静态滤镜
AliyunShaderFilterController *controller = [[editor getFilterManager] applyShadeFilterWithP
ath:path];
if (!controller) {
    // 添加LutFilter失败
}
// 2 删除静态滤镜
[[editor getFilterManager] removeFilter:controller];
```

动效滤镜

```
// 1 添加动效滤镜
AliyunEffectFilter *animationFilter = [[AliyunEffectFilter alloc] initWithFile:filterFolder
];
animationFilter.startTime = 2;
animationFilter.endTime = 10;
// ... 其他属性请参考接口文档
[editor applyAnimationFilter:animationFilter];
// 2 修改动效滤镜
animationFilter.startTime = 4;
animationFilter.endTime = 12;
// ... 其他属性请参考接口文档
[editor updateAnimationFilter:animationFilter];
// 3 删除动效滤镜
[editor removeAnimationFilter:animationFilter];
```

转场

- 转场支持自定义制作，制作方法请参见滤镜及转场。
- 目前短视频SDK提供的转场效果包括：AliyunTransitionEffectTypeCircle（圆形打开）、AliyunTransitionEffectTypeFade（淡入淡出）、AliyunTransitionEffectTypePolygon（五角星）、AliyunTransitionEffectTypeShuffer（百叶窗）、AliyunTransitionEffectTypeTranslate（平移）。接口参数请参考AliyunTransitionEffectType。

转场位置

转场需要设置在两个片段中间，所以如果只有一个视频片段，不能添加转场，转场位置从0开始，位置含义如下：

```
[----A视频段----] [----B视频段----] [----C视频段----]...[----N段视频----]
                 ^                 ^                  ^
 位置:           0                 1                 N-1 -->
```

设置转场

```
//添加
//注意：使用此接口前，需要先调用[editor stopEdit]，然后调用此接口，接着调用[editor startEdit]才会生效。
AliyunTransitionEffect *transition = [[AliyunTransitionEffect alloc] initWithPath:transitionFolder];
[editor applyTransition:transition atIndex:0];
//更新
//注意：转场的时长不能超过前后两段视频中最短的视频时长
transition.overlapDuration = 1;
// ... 其他更多属性请参考API文档
[editor updateTransition:transition atIndex:0];
//删除
[editor removeTransitionAtIndex:0];
```

MV

MV支持自定义制作，制作方法请参见MV。

```
// 添加MV
AliyunEffectMV *mv = [[AliyunEffectMV alloc] initWithFile:mvFolder];
[editor applyMV:mv];
// MV静音
[editor removeMVMusic];
// 删除MV
[editor removeMV];
```

时间特效

目前短视频SDK提供的时间特效包括：TimeFilterTypeSpeed（变速）、TimeFilterTypeRepeat（反复）、TimeFilterTypeInvert（倒放）。接口参数请参考TimeFilterType。

> **说明** 设置倒放时，如果视频的GOP过大(35)（可通过AliyunNativeParser来获取GOP），则需要先对视频做一次转码后（可通过视频裁剪来转码），再使用倒放特效。

```
// 1. 添加时间特效
// 1.1 变速
AliyunEffectTimeFilter *timeFilter = [[AliyunEffectTimeFilter alloc] init];
timeFilter.type = TimeFilterTypeSpeed;
timeFilter.param = 0.67;
timeFilter.startTime = 2;
timeFilter.endTime = 10;
// ...其他属性请参考API文档
[editor applyTimeFilter:timeFilter];
// 1.2 反复
AliyunEffectTimeFilter *timeFilter = [[AliyunEffectTimeFilter alloc] init];
timeFilter.type = TimeFilterTypeRepeat;
timeFilter.param = 3; // 例如重复3次
timeFilter.startTime = 2;
timeFilter.endTime = 10;
// ...其他属性请参考API文档
[editor applyTimeFilter:timeFilter];
// 1.3 倒放
AliyunEffectTimeFilter *timeFilter = [[AliyunEffectTimeFilter alloc] init];
timeFilter.type = TimeFilterTypeInvert;
// ...其他属性请参考API文档
[editor applyTimeFilter:timeFilter];
// 2. 删除时间特效
[editor removeTimeFilter:timeFilter];
```

设置音乐及音效

支持添加音乐和配音，给音频添加各种音效（淡入淡出效果、变声）。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

音乐

音乐分为背景音乐和配音。背景音乐不受时间特效影响（变速、重复、倒放等），而配音会受到时间特效的影响。

背景音乐

```
// 1. 添加背景音乐
AliyunEffectMusic *music =[[AliyunEffectMusic alloc] initWithFile:musicFilePath];
music.duration = 3;
music.audioMixWeight = 50; // 音量大小（混音权重）
// ... 其他更多属性请参考API文档
[editor applyMusic:music];
// 2. 删除背景音乐
[editor removeMusic:music];
```

配音

```
// 1. 添加配音
AliyunEffectDub *dub =[[AliyunEffectDub alloc] initWithFile:dubFilePath];
dub.startTime = 2;
dub.audioMixWeight = 50; // 音量大小（混音权重）
dub.audioDenoiseWeight = 50; // 降噪程度
// ... 其他更多属性请参考API文档
[editor applyDub:dub];
// 2. 删除配音
[editor removeDub:dub];
```

音效

- 淡入淡出：支持AliyunAudioFadeShapeLinear（线性曲线）、AliyunAudioFadeShapeSin（正弦函数曲线），接口参数请参考AliyunAudioFadeShape。
- 变声效果：接口参数请参考AliyunAudioEffectType。
  - AliyunAudioEffectLolita（萝莉）
  - AliyunAudioEffectUncle（大叔）
  - AliyunAudioEffectReverb（混响）
  - AliyunAudioEffectEcho（回声）
  - AliyunAudioEffectRobot（机器人）
  - AliyunAudioEffectBigDevil（大魔王）
  - AliyunAudioEffectMinions（小黄人）
  - AliyunAudioEffectDialect（方言）

淡入淡出

```
// 1. 添加音频前指定淡入效果（淡出类似，属性为fadeOut）
AliyunAudioFade *fadeIn = [[AliyunAudioFade alloc] init];
fadeIn.shape = AliyunAudioFadeShapeLinear;
fadeIn.duration = 2;
music.fadeIn = fadeIn; // 配音也一样设置
// 2. 添加音频后修改淡入效果（淡出类似，函数为 setAudioFadeOutShape:duration:streamId:）
[editor setAudioFadeInShape:AliyunAudioFadeShapeLinear duration:2 streamId:music.effectVid]
;
// 3. 添加音频后删除淡入效果（淡出类似，函数为 removeAudioFadeOutWithStreamId:）
[editor removeAudioFadeInWithStreamId:music.effectVid]; // 配音也一样调用
```

变声

```
// 1. 添加音频前设置变声
AliyunAudioEffect *audioEffect = [[AliyunAudioEffect alloc] init];
audioEffect.type = AliyunAudioEffectLolita;
audioEffect.weight = 50;
[dub.audioEffects addObject:audioEffect]; // 注意：暂时只支持添加一个
// 2. 添加音频后设置变声
[editor setAudioEffect:AliyunAudioEffectLolita weight:50 streamId:dub.effectVid];
// 3. 添加音频后删除变声
[editor removeAudioEffect:AliyunAudioEffectLolita streamId:dub.effectVid];
```

设置画中画

画中画功能允许在现用主轨道的基础上，添加一个或者多个画中画。

- 主轨道：编辑页面默认轨道，有且仅有一个主轨道，一个主轨道可以有多个视频流。
- 画中画：允许添加多个画中画，画中画允许设置位置，缩放，旋转等。创建画中画默认创建一个画中画轨道。画中画可以在不同画中画轨道中移动。

代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//获取画中画管理器，画中画管理器负责画中画的增删改查
AliyunPipManager * pipManager = [editor getPipManager];
//添加画中画
// 1. 在最上层添加一个画中画轨道，把画中画片段添加进该轨道
// 1.1 添加视频片段
NSError *error = nil;
AliyunPipClipController *pipClipController = [pipManager addClipWithType:AliyunPipClipTypeV
ideo path:xxVideoPath error:&error];
// 1.2 添加图片片段
NSError *error = nil;
AliyunPipClipController *pipClipController = [pipManager addClipWithType:AliyunPipClipTypeI
mage path:xxImagePath error:&error];
// 2. 把画中画插入到指定一个画中画轨道
// 2.1 创建画中画片段
AliyunPipClip *pipClip = [[AliyunPipClip alloc] initWithClipType:AliyunPipClipTypeVideo cli
pPath:xxxVideoPath];
// ... 更多画中画片段设置请参考文档：https://alivc-demo-cms.alicdn.com/versionProduct/doc/short
Video/iOS_cn/Classes/AliyunPipClip.html
// 2.2 插入到指定的轨道中
NSError *error = nil;
AliyunPipClipController *pipClipController = [pipManager addClipWithModel:pipClip toTrack:p
ipManager.trackControllers.firstObject error:&error]; // 例如添加到第一条轨道里
//删除画中画
NSError *error = nil;
[pipManager removePipClipController:pipClipController error:&error];
//切换画中画轨道
[pipManager movePipClipController:pipController toTrack:pipManager.trackControllers.firstOb
ject withStartTime:0]; // 例如移动到第一条轨道的开始位置
//修改画中画片段
// 例如直接修改画中画位置
pipController.clip.center = CGPointMake(100, 100);
// 例如批量修改位置、大小、旋转等
[pipController beginEdit];
pipController.clip.center = CGPointMake(100, 100);
pipController.clip.scale = 0.7;
pipController.clip.rotation = M_PI_2;
[pipController endEdit];
//点击测试
// 例如获取当前时间的某个触摸点最上层的画中画片段
double currentTime = [[editor getPlayer] getCurrentTime];
AliyunPipClipController *pipClipController = [pipManager hitTest:touchPoint withTime:curren
tTime];
```

设置字幕及贴纸

- 字幕及贴纸统一通过管理器 AliyunStickerManager进行管理，字幕及贴纸本身的状态分别通过字幕控制器AliyunCaptionStickerController和贴纸控制器AliyunStickerController进行管理。而字幕控制器和渲染控制器都继承于基础渲染控制器AliyunRenderBaseController，所以修改字幕或贴纸也采用修改基础元素属性同样的逻辑。
- 基于字幕，短视频SDK还提供了花字及气泡文字的特效。花字和气泡文字支持自定义制作，制作的规范及方法请参见花字和动图。
- 贴纸分为静态贴纸和动态贴纸，动态贴纸和气泡文字类似，只不过缺少了文字部分。动态贴纸支持自定义制作，制作规范及方法请参见动图。
- 代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

管理器

- 管理器负责字幕的及贴纸的增删改查，接口参数请参考AliyunStickerManager。
- 字幕和贴纸都继承于渲染模型AliyunRenderModel，有基础渲染元素的属性，您可以通过基于预览坐标系的点击位置来查找某一时刻在最上层的字幕或贴纸的控制器。

```
//获取管理器
AliyunStickerManager *stickerManager = [editor getStickerManager];
//添加
// 例如添加字幕
AliyunCaptionStickerController *captionController = [stickerManager addCaptionText:@"Hello"
bubblePath:nil startTime:0 duration:5];
//删除
[stickerManager remove:gifController];
//查找
// 例如获取当前时间的某个触摸点最上层的字幕或贴纸
double currentTime = [[editor getPlayer] getCurrentTime];
AliyunRenderBaseController *controller = [stickerManager findControllerAtPoint:touchPoint a
tTime:currentTime];
```

字幕

```
//添加字幕
AliyunCaptionStickerController *captionController = [stickerManager addCaptionText:@"Hello"
bubblePath:nil startTime:0 duration:5];
//修改字幕
[captionController beginEdit];
captionController.model.text = xxx;
captionController.model.outlineWidth = 3;
captionController.model.outlineColor = UIColor.redColor;
// ... 其他属性修改请参考AliyunCaptionSticker接口文档
[captionController endEdit];
```

花字

```
// 应用花字
captionController.model.fontEffectTemplatePath = fontEffectFolder; // 花字特效资源包文件夹目录
// 取消花字
captionController.model.fontEffectTemplatePath = nil;
```

文字气泡

```
// 应用气泡
captionController.model.resourePath = bubbleEffectFolder; // 文字气泡特效资源包文件夹目录
// 取消气泡
captionController.model.resourePath = nil;
```

动态贴纸

```
//添加动态贴纸
AliyunGifStickerController *gifController = [stickerManager addGif:gifFilePath startTime:0
duration:5];
//修改动态贴纸
[gifController beginEdit];
gifController.gif.center = xxx;
// ...其他属性修改请参考AliyunGifSticker接口文档
[gifController endEdit];
```

静态贴纸

```
//添加静态贴纸
AliyunImageStickerController *imageController = [stickerManager addImage:imageFilePath star
tTime:0 duration:5];
//修改静态贴纸
[imageController beginEdit];
imageController.image.center = xxx;
// ...其他属性修改请参考AliyunImageSticker接口文档
[imageController endEdit];
```

草稿箱

每次编辑会生成一个编辑任务，以工程配置的形式记录下来。如果您不想马上导出当前编辑结果，可以将编辑状态保存为草稿，下次通过草稿加载就能够恢复上次的编辑状态继续编辑。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

工程配置

编辑状态是以草稿对象的形式，以时间线的结构来描述的，每一个编辑动作最终会体现到这个工程配置上的状态改变。建议您将工程配置跟您的编辑界面相关联。草稿恢复时使用该工程配置恢复您的编辑状态对应的界面状态。

```
// 编辑中获取工程配置对象
AliyunEditorProject *project = [editor getEditorProject];
```

草稿对象

编辑状态可以保存为一个草稿对象AliyunDraft。您可以按以下方式保存当前的编辑状态。

```
// 0. 保存为草稿前，您需要定义草稿应该保存在哪里，我们提供了一个草稿管理对象作为草稿存储管理的容器
AliyunDraftManager *draftManager; // 草稿管理的介绍请参考下方文档说明
// 1. 指定草稿的标题保存，需要指定保存到哪里，返回草稿对象
AliyunDraft *draft = [editor saveToDraft:draftManager withTitle:@"your draft title"];
// 2.1 不指定草稿标题（使用上一次草稿标题）
AliyunDraft *draft = [editor saveToDraft:draftManager];
// 2.2 可以在获得草稿对象后修改标题
[draft renameTitle:@"your draft title"];
```

为了能更好地标识一个草稿，助力您开展本地或云端草稿业务，短视频SDK预留了一个ID用于标识一个草稿，您可以通过接口设置为您业务标识的ID。

```
[draft changeProjectId:@"your custom project id"];
```

草稿封面

编辑过程中，会自动生成一个合适的封面图（通常为视频的第一帧画面）。如果您觉得不合适，也可以通过接口替换为自定义的封面。

```
[editor updateCover:yourCoverImage];
// 也可以通过设置为空，让我们为您自动生成（默认）
[editor updateCover:nil];
```

在获得草稿对象后也支持更换封面图。

```
[draft updateCover:yourCoverImage];
```

说明 如果需要自定义草稿封面，建议尽早设置，因为指定为自定义草稿封面后将不再自动去更新封面内容，因此，尽早设置后会减少不必要的更新动作，从而提高编辑性能。

草稿管理

注意 由于草稿管理在初始化时会解析出所管理的所有草稿对象，所以有一定的性能损耗，建议作为一个全局的对象不要频繁地创建和销毁。

```
//初始化
//为更好地实现用户隔离，以ID作为草稿管理的标识，建议您使用用户的标识作为草稿管理的标识
NSString *draftManagerId;
//实例化
AliyunDraftManager *draftManager = [[AliyunDraftManager alloc] initWithId:draftManagerId];
//草稿列表
// 获取草稿列表
NSArray<AliyunDraft *> *draftList = draftManager.draftList;
// 也可以监听草稿列表的变化
draftManager.delegate = yourListener;
// 实现 AliyunDraftManagerDelegate协议 即可
// - (void) onAliyunDraftManager:(AliyunDraftManager *)mgr listDidChange:(NSArray<AliyunDra
ft *> *)list;
//删除草稿
[draftManager deleteDraft:targetDraft];
//复制草稿
[draftManager copyDraft:fromDraft toPath:newDraftTaskPath withTitle:@"your new draft title"
];
```

草稿加载

由于编辑过程可能会用到多种资源，出于性能和存储空间的考虑，我们一般不会把用到的资源都拷贝到任务目录下，因此在从草稿恢复编辑状态前我们需要确保此次编辑中用到的资源都处于准备好的状态。所以，在草稿恢复到编辑状态前，您必须要处理一些资源的加载任务，以确保编辑用到的资源都已经准备好了。以下两类资源建议您特别关注：

- 相册中的媒体资源：需要确保开始编辑之前拥有读取的权限。
- 动态自定义的字体资源：需要确保开始编辑之前对应的字体已经注册到系统里，至少注册到当前app会话中。

```
[draft load:^(NSArray<AliyunDraftLoadTask *> *tasks) {
    for (AliyunDraftLoadTask *task in tasks) {
        // 资源加载任务处理
        // 1. 获取当前任务的资源模型
        AEPResourceModel *resource = task.resource;
        // 2. 处理...
        // 更多资源类型的处理可以参考官方Demo里的 AliyunDraftLoader.m
        // 3. 标记处理结果
        // 3.1.1 成功：忽略结果
        [task onIgnore];
        // 3.1.2 成功：需要修改资源模型的属性
        AEPSource *resultSoruce = [resource.source createWithPath:@"result path"]; // 例如加载后资源路径发生了改变
        [task onSuccess:resultSoruce];
        // 3.2.1 失败：把对应的节点删除掉继续加载
        [task onFailToRemove]; // 例如某个字幕的字体加载不了，选择把该字幕删除掉继续加载打开
        // 3.2.2 失败：标记整体加载失败，把下面的error作为整体的加载失败原因
        NSError *error = xxxx; // 需要您给出具体加载失败的原因
        [task onFailToStopWithError:error]; // 发生这种失败时，建议您主动停止其他的加载任务
    }
} completion:^(NSString *taskPath, AliyunEditorBaseProject *project, NSError *error) {
    if (!taskPath || !project || error) {
        // 加载失败处理...
        // 一般情况下，error.localizedDescription中会有详细的失败原因
        return;
    }
    // 加载成功
    // 可以使用 taskPath创建Editor以获得恢复后的编辑状态，请参考编辑视频的初始化文档
}];
```

草稿上传

编辑状态主要由工程配置和资源组成。所以只要把这两部分同步到云端即可实现云草稿。接口参数请参考AliyunDraftProjectUploadTask。

为了性能和存储空间，默认不会拷贝编辑用到的所有资源，但草稿会记录所有用到资源的描述。为了更好地索引资源，提供了AEPSource.path（本地路径）、AEPSource.sourceId（资源ID）等多种方式来描述一个资源，详细可参考 AEPSource。在草稿资源的加载、上传、下载等过程中，除了会提供资源的描述，还会提供当前加载的该资源所在的节点对象、时间线所属模块等信息，详细可参考AEPResourceModel。

草稿上传分为如下两个过程：

1. 上传编辑中所有用到的资源，这个过程中会修改工程配置中资源的描述。
2. 上传资源描述修改后的工程配置文件。

```
[draft uploadWithResourceUploader:^(NSArray<AliyunDraftLoadTask *> *tasks) {
    for (AliyunDraftLoadTask *task in tasks) {
        // 上传任务处理
        // 1. 获取当前任务的资源模型
        AEPResourceModel *resource = task.resource;
        // 2. 处理...
        // 更多资源类型的处理可以参考官方Demo里的 AliyunDraftLoader.m
        // 3. 标记处理结果
        // 3.1.1 成功：忽略结果
        [task onIgnore]; // 例如业务内置资源不需要上传处理
        // 3.1.2 成功：需要修改资源模型的属性
        AEPSource *resultSoruce = [resource.source createWithURL:@"resource URL"]; // 例如上传资源后获得网络链接
        [task onSuccess:resultSoruce];
        // 3.2.1 失败：把对应的节点删除掉继续上传
        [task onFailToRemove]; // 例如某个字幕的字体上传失败了，选择把该字幕删除掉继续上传其他的
        // 3.2.2 失败：标记整体上传失败，把下面的error作为最后的整体上传失败原因
        NSError *error = xxx; // 需要您给出具体上传失败的原因
        [task onFailToStopWithError:error]; // 发生这种失败时，建议您主动停止其他的上传任务
    }
} projectUploader:^(AliyunDraftProjectUploadTask *projTask) {
    // 添加云草稿处理
    // 1.1 获取当前草稿的工程配置文件路径
    NSString *projectFilePath = projTask.projectFilePath;
    // 1.2 处理工程配置文件
    NSString *projectUrl = [yourUploader upload:projectFilePath]; // 例如上传配置文件，返回网络路径
    // 2.1 同步云草稿
    NSString *projectId = [yourApi addCloudDraft:projectUrl]; // 例如调用您的业务服务返回一个标识ID
    // 2.2 更新草稿的工程ID
    [projTask.draft changeProjectId:projectId];
    // 3. 标记处理结果
    // 3.1 成功
    [projTask onSuccess];
    // 3.2 失败
    NSError *error = xxx; // 需要您给出添加失败的原因
    [projTask onFailWithError:error];
} completion:^(NSError *error) {
    if (error) {
        // 上传失败处理...
        return;
    }
    // 上传成功处理...
}];
```

草稿下载

因为草稿的工程配置记录着所有用到的资源描述，所以只要提供工程配置文件就能枚举出所有的资源，把资源同步到本地就能完成草稿的下载。

跟保存草稿一样，需要确定草稿保存到哪里，所以下载接口定义在了本地草稿管理器AliyunDraftManager。

```
// 0. 通过您的业务服务获取云端草稿对应的工程配置文件和工程ID
NSString *projectFilePath = xxx;
NSString *projectId = xxx;
// 1. 下载草稿
[draftManager downloadDraftWithProjectFile:projectFilePath  resourceDownloader:^(NSArray<Al
iyunDraftLoadTask *> *tasks) {
    for (AliyunDraftLoadTask *task in tasks) {
        // 下载任务处理
        // 1. 获取资源
        AEPResourceModel *resource = task.resource;
        // 2. 处理...
        NSString *localPath = [yourDownloader download:resource.source.URL]; // 例如网络下载
        // 3. 标记处理结果
        AEPSource *localSource = [resource.source createWithPath:localPath]; // 例如下载到本
地了
        [task onSuccess:localSource];
        // 更多结果标记参考 加载任务
    }
} completion:^(AliyunDraft *draft, NSError *error) {
    if (error || !draft) {
        // 下载失败处理...
        return;
    }
    // 下载成功处理...
    [draft changeProjectId:projectId]; // 建议同步一下云端的ID
    // 其他更多处理...
}];
```

其他设置

涂鸦

短视频SDK封装了一套涂鸦接口，包含画板、画笔等，整个涂鸦操作由涂鸦画布视图（AliyunICanvasView）完成。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 1. 创建画笔
AliyunIPaint *paint = [[AliyunIPaint alloc]initWithLineWidth:5.0 lineColor:UIColor.whiteCol
or];
// 2. 添加涂鸦视图
AliyunICanvasView *paintView = [[AliyunICanvasView alloc]initWithFrame:CGRectMake(0, 0, 100
, 100) paint:paint];
[yourView addSubview:paintView];
// 3. 涂鸦
// 触摸视图即能进行涂鸦
// 4. 其他操作
// 4.1 撤销一步
[paintView undo];
// 4.2 反撤销一步
[paintView redo];
// 4.3 撤销本次涂鸦所有的操作
[paintView undoAllChanges];
// 4.4 清空所有线条(不可恢复)
[paintView remove];
// 5. 完成涂鸦
UIImage *image = [paintView complete];
NSString *paintPath = xxx;
[UIImagePNGRepresentation(image) writeToFile:paintPath atomically:YES];
// 6. 添加到编辑
AliyunEffectImage *paintImage = [[AliyunEffectImage alloc] initWithFile:paintPath];
[editor applyPaint:paintImage linesData:paintView.lines];
// 7. 删除涂鸦
[editor removePaint:paintImage];
```

5.6.4. 导出视频

本文为您介绍iOS端短视频SDK导出视频的方法。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

名称	功能
AliyunIExporter	导出协议，用于获取导出控制器，配置导出参数，开始导出视频。
AliyunIExporterCallback	导出状态回调协议，用于设置导出完成、导出进度、导出失败等回调。

名称	功能
AliyunVideoParam	视频参数设置类，设置导出视频的编码类型、帧率等参数。

导出控制

初始化AliyunIExporter类，获取导出控制器，配置导出参数，开始导出视频。

接口参数请参考AliyunIExporter及AliyunVideoParam。

```
// 获取导出控制器
id<AliyunIExporter> exporter = [editor getExporter];
// 设置导出参数
AliyunVideoParam *param = [[AliyunVideoParam alloc] init];
param.fps = 60;
param.gop = 250;
param.videoQuality = AliyunVideoQualityHight;
// ... 其他更多导出属性请参考接口文档
[exporter setVideoParam:param];
// 设置输出视频水印
AliyunEffectImage *watermark = [[AliyunEffectImage alloc] initWithFile:watermarkPath];
watermarkPath.frame = CGRectMake(10, 10, 50, 50);
// .. 其他更多属性请参考接口文档
[exporter setWaterMark:watermark];
// 设置片尾水印
AliyunEffectImage *tailWatermark = [[AliyunEffectImage alloc] initWithFile:watermarkPath];
tailWatermark.frame = CGRectMake(10, 10, 50, 50);
tailWatermark.endTime = 2;
// .. 其他更多属性请参考接口文档
[exporter setTailWaterMark:tailWatermark];
// 开始导出
[exporter startExport:outputPath];
// 暂停导出
[exporter pauseExport];
// 恢复继续导出
[exporter resumeExport]
// 取消导出
[exporter cancelExport];
```

设置回调

通过设置回调，及时获取视频导出的进展和状态。接口参数请参考AliyunIExporterCallback。

```
// 设置导出状态回调
editor.exporterCallback = self;
// 状态协议：AliyunIExporterCallback
- (void)exporterDidEnd:(NSString *)outputPath {
    // 导出结束回调
}
- (void)exporterDidCancel {
    // 导出取消回调
}
- (void)exportProgress:(float)progress {
    // 导出进度回调
}
- (void)exportError:(int)errorCode {
    // 导出错误回调
}
```

5.7. 视频模板（剪同款）

视频模板功能即通过替换模板中的视频、图片、文字，导出生成新的视频，是一种快速或规模化的视频生产方式。可通过短视频的模板能力实现剪同款功能。

版本支持

版本	是否支持
专业版	支持 **说明** 短视频SDK V3.27及以上版本才支持视频模板功能。
标准版	不支持
基础版	不支持

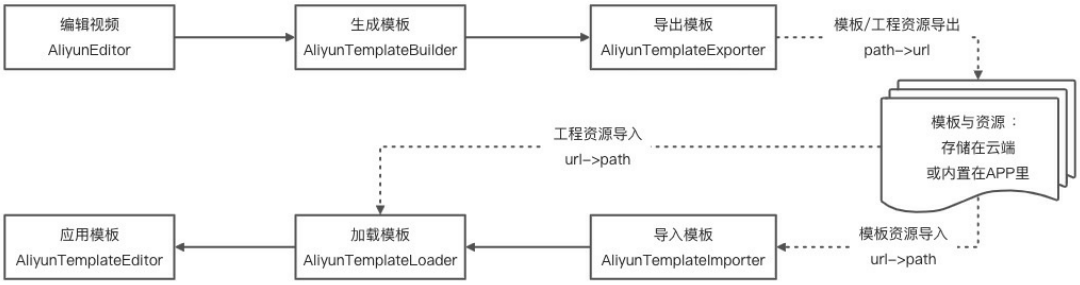
相关类功能

类名	功能
AliyunTemplate	模板。
AliyunTemplateParam	模板参数。
AliyunTemplateNode	模板编辑应用节点基类。
AliyunTemplateClipNode	模板片段编辑应用节点。
AliyunTemplateCaptionNode	模板字幕编辑应用节点。
AliyunTemplateEditor	模板应用编辑器。

类名	功能
AliyunTemplateModifyContent	定义模板生成器中要修改的内容。
AliyunTemplateBuilder	模板生成器。
AliyunTemplateExporter	模板导出器。
AliyunTemplateLoader	模板加载器。
AliyunTemplateImporter	模板导入器。
ATResourceModel	模板资源模型。
AliyunTemplateResourceExport	模板资源导出管理器。
AliyunTemplateResourceImport	模板资源导入管理器。

视频模板应用流程

视频模板的使用流程如下图所示：



生成模板

可以对编辑器或草稿目录生成模板文件，并存储在指定的目录下；也可以对已经生成的模板文件进行打开编辑。无论哪种形式，最终可以获取AliyunTemplate对象，并提供更新封面、预览视频、预览标题、预览模板ID、锁定节点等接口。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 生成模板文件，并进行更新
NSString *taskPath = [[AlivcTemplateResourceManager builtTemplatePath] stringByAppendingPathComponent:[NSUUID UUID].UUIDString];
AliyunTemplateBuilder *builder = [AliyunTemplateBuilder build:taskPath editorTaskPath:_taskPath];
if (builder) {
    // update title
    [builder updateTitle:_titleView.text];
    // update cover url
    NSString *coverPath = [_taskPath stringByAppendingPathComponent:@"template_cover.png"];
    NSData *data = UIImagePNGRepresentation(_coverImage);
    [data writeToFile:coverPath atomically:YES];
    [builder updateCover:coverPath];
    // update preview with remote url
    [builder updatePreviewVideo:_outputPath];
    // save all
    [builder save];
    AlivcTemplateBuilderViewController *vc = [[AlivcTemplateBuilderViewController alloc] initWithEditorTaskPath:taskPath isOpen:YES];
    [self.navigationController pushViewController:vc animated:YES];
    return;
}
```

导出模板

生成的模板，其关联到的自定义资源（包括模板资源和工程资源，APP内置资源或平台资源除外）都是以绝对路径（Source.Path）存在配置文件中。如果模板需要在不同的设备或平台中使用，那么需要进行导出模板。通过导出模板，把本地关联的资源进行拷贝到相对目录或上传，并返回指定的地址（Source.URL），从而达到跨设备使用目的。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
NSString *taskPath = @"存储目录";
NSString *templateTaskPath = @"模板目录";
AliyunTemplateResourceExport *resourceExport = [AliyunTemplateResourceExport new];
// 模板资源导出，目的是把本地自定义资源的路径转换为http（需要把本地资源进行上传）或自定义协议地址（例如本地相对"alivc_resource://relation?path="），参考demo
resourceExport.selfResourceExport = [AlivcTemplateResourceManager templateResourceExport:taskPath];
// 工程资源导出，目的是把本地自定义资源的路径转换为http（需要把本地资源进行上传）或自定义协议地址（例如本地相对"alivc_resource://relation?path="），参考demo
resourceExport.projectResourceExport = [AlivcTemplateResourceManager projectResourceExport:taskPath];
// 执行导出
[AliyunTemplateExporter export:taskPath templateTaskPath:templateTaskPath resourceExport:resourceExport completed:^(NSError *error) {
    if (error) {
        // 失败
    }
    else {
        // 成功
    }
}];
```

导入模板

APP中需要使用的模板，往往来自内置或网络下载，这些模板通常都能够跨设备使用，当需要在当前设备使用模板时，需要通过导入来处理其关联的资源。通过导入模板，把非本地关联的资源进行下载（必要时），最终把自定义协议地址的资源进行处理，并返回指定的绝对路径（Source.Path），这样在加载模板时便确保能访问到对应资源。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
NSString *tempTaskPath = @"模板目录";
NSString *taskPath = @"存储目录";
// 模板资源导入，目的是把自定义协议地址（例如本地相对”alivc_resource://relation?path=“）转换为本地全路径，详细可参考demo代码示例
AliyunResourceImport *resourceImport = [AlivcTemplateResourceManager templateResourceImport:tempTaskPath reset:YES];
[AliyunTemplateImporter import:taskPath templateTaskPath:tempTaskPath resourceImport:resourceImport completed:^(NSError *error) {
    if (error) {
        // 出错，删除本地模板文件，下次可以再次尝试执行导入
        [[NSFileManager defaultManager] removeItemAtPath:taskPath error:nil];
    }
    else {
        // 成功
    }
}];
```

加载模板

对已导入的模板，需要通过加载后才能使用，对Project资源进行导入后才能应用模板。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 加载Loader
NSString *taskPath = @"模板目录";
AliyunTemplateLoader *loader = [[AliyunTemplateLoader alloc] initWithTaskPath:taskPath];
if (loader) {
    self.loader = loader;
}
else {
    // 失败
}
// 导入Project资源后，应用模板
NSString *templateTaskPath = @"模板目录";
__weak typeof(self) weakSelf = self;
[self.loader loadProject:[AlivcTemplateResourceManager projectResourceImport:templateTaskPath reset:NO shouldDownload:YES] completed:^(NSError *error) {
    if (error) {
        // 加载资源失败
    }
    else {
        // 成功，应用模板
        AlivcTemplateEditorViewController *tevc = [[AlivcTemplateEditorViewController alloc] initWithTemplateTaskPath:templateTaskPath];
        [weakSelf.navigationController pushViewController:tevc animated:YES];
    }
}];
```

应用模板

可以对指定的模板，或者一个已经应用模板的草稿进行编辑，可预览或替换视频、图片、字幕等节点，并导出视频。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
// 创建应用模板编辑器
if (self.templateTaskPath) {
    // 新建
    NSString *taskPath = [[AlivcTemplateResourceManager projectTemplatePath] stringByAppend
ingPathComponent:[NSUUID UUID].UUIDString];
    self.aliyunEditor = [AliyunTemplateEditor createTemplateEditor:self.templateTaskPath on
TaskPath:taskPath];
}
else if (self.taskPath) {
    // 打开
    self.aliyunEditor = [AliyunTemplateEditor openTemplateEditor:self.taskPath];
}
// 设置预览视图
self.aliyunEditor.preview = self.videoDisplayView;
// 加载编辑器，加载后才能进行后续操作
int ret = [self.aliyunEditor loadEditor];
if (ret != ALIVC_COMMON_RETURN_SUCCESS) {
    NSLog(@"加载失败，可能是授权问题：%d", ret);
}
// 开始编辑
[self.aliyunEditor startEdit];
// 播放
[self.aliyunEditor..getPlayer play];
// 获取更换片段节点列表，按照锁定、开始时间进行排序
NSArray<AliyunTemplateClipNode *> *sortNodes = [[self.aliyunEditor clipNodes] sortedArrayUs
ingComparator:^NSComparisonResult(AliyunTemplateClipNode *obj1, AliyunTemplateClipNode *obj
2) {
    if (obj1.lock && !obj2.lock) {
        return NSOrderedAscending;
    }
    else if (!obj1.lock && obj2.lock) {
        return NSOrderedDescending;
    }
    else if (obj1.timelineIn < obj2.timelineIn) {
        return NSOrderedAscending;
    }
    return NSOrderedDescending;
}];
// 更换视频/图片片段
int ret = [self.aliyunEditor updateClipNode:node clipPath:sourcePath clipType:AliyunClipVid
eo];
if (ret == ALIVC_COMMON_RETURN_SUCCESS) {
    // 成功
}
else {
    NSLog(@"替换失败：%d", ret);
}
// 导出
AlivcExportViewController *controller = [[AlivcExportViewController alloc] init];
```

```
controller.taskPath = self.aliyunEditor.taskPath;
controller.outputPath = [[AlivcTemplateResourceManager applyTemplatePath] stringByAppending
PathComponent:fileName];
controller.outputSize = self.aliyunEditor.getEditorProject.config.outputResolution;
controller.backgroundImage = [UIImage imageWithContentsOfFile:self.aliyunEditor.getCurrentT
emplate.cover.path];
controller.coverImage = controller.backgroundImage;
controller.draft = draft;
[self.navigationController pushViewController:controller animated:YES];
```

5.8. 视频拼接

短视频SDK提供了视频拼接接口AliyunIMixComposer。该接口实现离线多画面合并成一个视频的功能，例如画中画、九宫格、左右分屏、上下分屏等视频效果，支持添加多轨道视频。本文为您介绍iOS端短视频SDK视频拼接的流程以及示例代码。

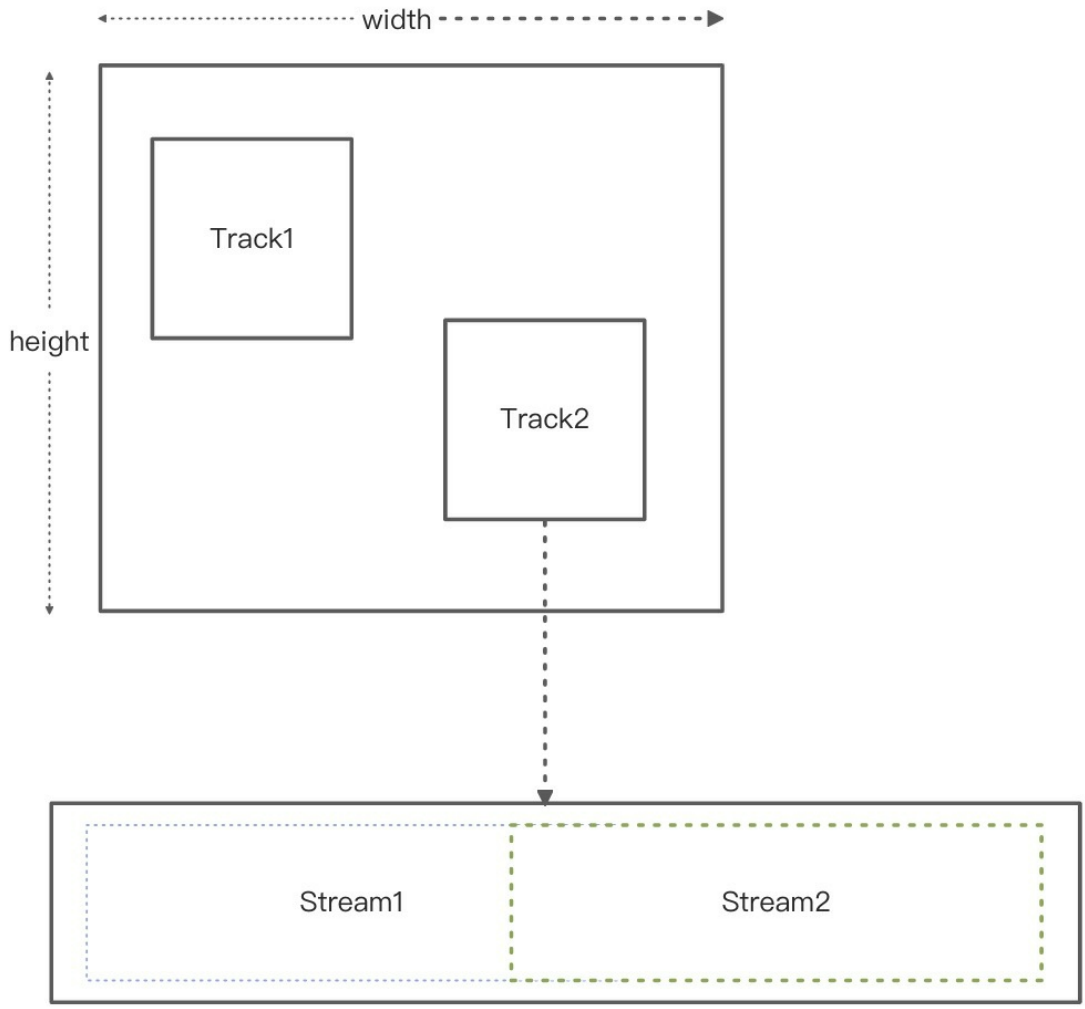
版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

类名	功能
AliyunMixComposer	拼接功能核心类，提供了对多个视频进行拼接的能力，属于底层实现类，能够实现多视频同画面效果，比如前后拼接、左右分屏、画中画、九宫格等效果。
AliyunMixTrack	视频合并轨道，由AliyunMixComposer创建，可添加视频流。
AliyunMixStream	视频流，具体用来合成拼接的视频，并加入到视频轨道中。
AliyunPureColorBorderInfo	轨道流边框信息。

拼接结构图



拼接流程

阶段	流程	说明	示例代码
基础	1	创建拼接实例，设置代理回调。	初始化
	2	创建多个轨道，创建视频流，再将视频流分别添加到各轨道上。	创建轨道
	3	配置视频拼接后的输出路径、视频宽高等参数。	配置输出参数
	4	设置回调，开始拼接。	开始拼接
进阶	5	取消、暂停、继续拼接，按需设置。	拼接控制

初始化

创建拼接实例，设置代理回调。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//创建mixComposer
AliyunMixComposer *mixComposer = [[AliyunMixComposer alloc] init];
//设置代理回调
mixComposer.delegate = self;
```

创建轨道

创建多个轨道，创建视频流，再将视频流分别添加到各轨道上。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

添加轨道

```
//添加一个在左边的轨道
AliyunMixTrack *recordTrack = [mixComposer createTrack:CGRectMake(0,0,360,720)];
//添加一个在右边的轨道
AliyunMixTrack *playerTrack = [mixComposer createTrack:CGRectMake(360,0,360,720)];
//设置轨道参数
//指定音频输出的轨道
recordTrack.outputAudioReferenceTrack = YES;
//指定音频输出占比
recordTrack.outputAudioWeight = 100;
//指定轨道作为输出视频的时长
recordTrack.outputDurationReferenceTrack = YES;
//设置轨道边框
AliyunPureColorBorderInfo *info = [[AliyunPureColorBorderInfo alloc] init];
info.width = 10.f;
info.cornerRadius = 10.f;
info.color = [UIColor redColor];
recordTrack.borderInfo = info;
```

> **说明** 可以通过设置每个轨道的outputAudioReferenceTrack及outputAudioWeight指定输出视频的音频使用。视频的时长的设置，如果轨道都设置了outputDurationReferenceTrack = YES，以轨道的创建先后为准，后创建的会覆盖前一个。

给轨道添加视频流

```
//添加视频流到左边轨道
AliyunMixStream *recordStream = [[AliyunMixStream alloc] init];
recordStream.filePath = videoPath;
recordStream.mode = AlivcContentModeScaleAspectFit;
[recordTrack addStream:recordStream];
//添加视频流到右边轨道
AliyunMixStream *playerStream = [[AliyunMixStream alloc] init];
playerStream.filePath = mixVideoFilePath;
playerStream.mode = AlivcContentModeScaleAspectFit;
[playerTrack addStream:playerStream];
```

配置输出参数

配置视频拼接后的输出路径、视频宽高等参数。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//输出路径
mixComposer.outputPath = self.outputPath;
//输出分辨率
mixComposer.outputSize = CGSizeMake(720,720);
//输出帧率
mixComposer.fps = 30;
//关键帧间隔
mixComposer.gop = 90;
//视频质量
mixComposer.videoQuality = AliyunVideoQualityHight;
```

开始拼接

开始拼接视频，并代理回调。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//开始拼接
[mixComposer start];
//代理回调
//拼接进度
- (void)mixComposerOnProgress:(float)progress {
}
//拼接完成
- (void)mixComposerDidComplete {
}
//拼接出错
- (void)mixComposerDidError:(int)error {
}
```

拼接控制

按需设置取消、暂停、继续拼接。代码中需要使用的参数详情，请参考接口文档。接口链接请参见相关类功能。

```
//暂停拼接
[mixComposer pause];
//继续拼接
[mixComposer resume];
//取消拼接
[mixComposer cancel];
```

5.9. 视频上传

本文为您介绍iOS端短视频SDK视频上传的功能以及流程说明。

功能介绍

视频点播支持通过多种方式上传媒体文件（音频、视频、图片等）到点播存储，详细请参见媒体上传。其中，上传SDK提供了一套单独进行合成上传的功能接口，用来实现将编辑完的视频在另一界面合成上传，核心类AliyunVodPublishManager封装了视频合成与上传功能，方便客户端更好地合成与上传视频。阿里云短视频SDK通过调用核心类AliyunVodPublishManager以完成视频的上传。

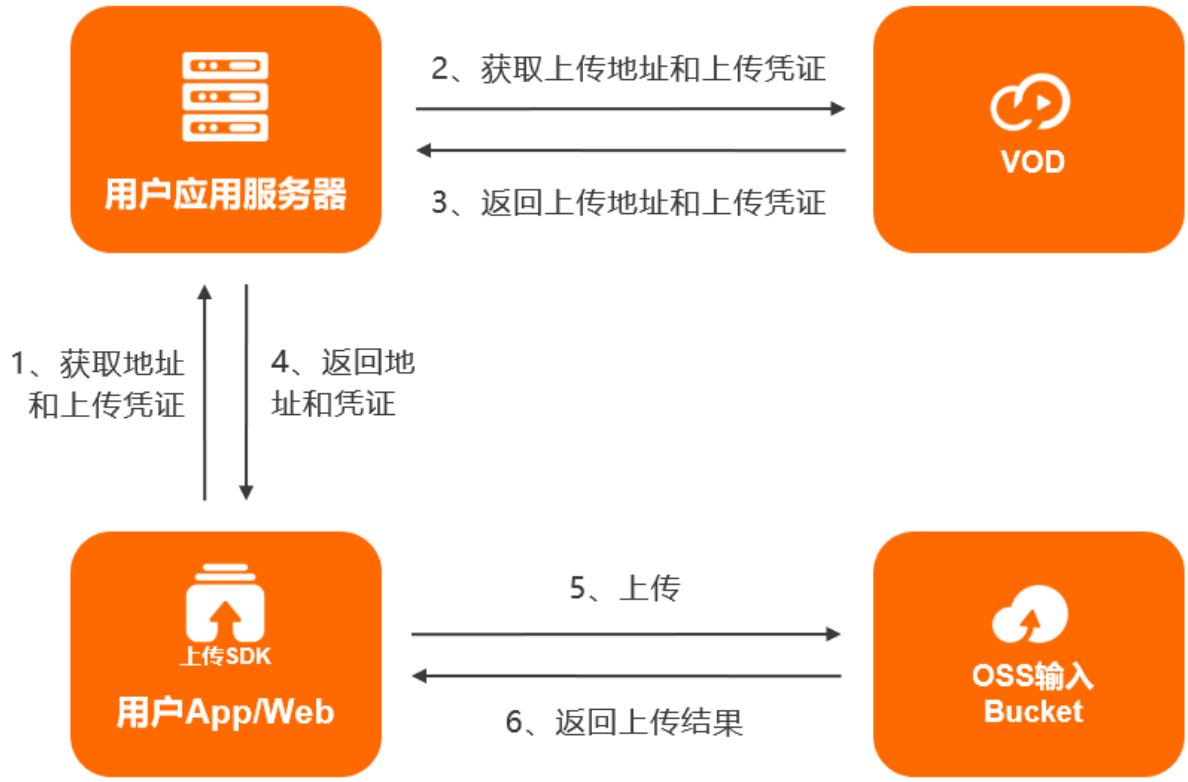
版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	不支持

相关类功能

类名	功能
AliyunVodPublishManager	上传功能核心类，包括初始化、上传视频或图片、更新上传凭证等核心上传功能。

视频上传流程



1. 当客户端（用户APP/Web端）通过上传SDK上传视频或图片时，首先向用户应用服务器发起请求，获取上传地址和凭证。
2. 应用服务器通过OpenAPI，向视频点播服务发起CreateUploadVideo（视频）或CreateUploadImage（图片）请求。

3. 请求成功，则视频点播服务将返回上传地址、上传凭证、VideoId（视频）、ImageURL（图片）给用户应用服务器。
4. 用户应用服务器返回上传地址和上传凭证给客户端。
5. 客户端添加本地文件，并设置上传地址和凭证，开始上传到OSS Bucket。
6. OSS返回上传结果。

初始化

初始化上传实例，设置上传回调。

```
_vodManager = [AliyunVodPublishManager new];
//设置上传回调
_vodManager.uploadCallback = self;
```

获取上传地址及凭证

AliyunVodPublishManager是通过上传地址和上传凭证上传，在上传之前，需要获取上传地址及上传凭证，详细操作请参见获取音视频上传地址和凭证和获取图片上传地址和凭证。

> **说明** 此处只是在获取上传地址和凭证，并未开始上传，获取到上传地址和凭证后，需要再执行上传。

上传

上传视频、图片等媒体文件。

```
//通过调用获取视频上传地址和凭证CreateUploadVideo或获取图片上传地址和凭证CreateUploadImage接口后将返回以下参数用于上传视频或图片
//videoPath:视频文件地址，请与获取上传地址和凭证时输入的视频文件地址保持一致
//imagePath:图片文件地址，请与获取上传地址和凭证时输入的视频文件地址保持一致
//uploadAddress:上传地址
//uploadAuth:上传凭证，凭证过期后请参考下文刷新上传凭证
//上传视频
int ret = [_vodManager uploadVideoWithPath:videoPath uploadAddress:uploadAddress uploadAuth
:uploadAuth];
//上传图片
int ret = [_vodManager uploadImageWithPath:imagePath uploadAddress:uploadAddress uploadAuth
:uploadAuth];
```

刷新上传凭证

因上传凭证带有时效性（UploadAuth字段的Expiration变量赋予了上传授权的过期时间，默认有效期为3000秒），所以当上传凭证过期后，需要在过期回调onUploadTokenExpired方法中重新获取上传凭证上传，更多信息，请参考刷新视频上传凭证。

```
//刷新上传凭证，uploadAuth:上传凭证
[AliyunVodPublishManager refreshWithUploadAuth:vodUploadAuth];
```

上传回调

```
//上传成功回调
- (void)publishManagerUploadSuccess:(AliyunVodPublishManager *)manager {
}
//上传失败回调
- (void)publishManager:(AliyunVodPublishManager *)manager uploadFailedWithCode:(NSString *)
code message:(NSString *)message {
}
//上传进度回调
- (void)publishManager:(AliyunVodPublishManager *)manager uploadProgressWithUploadedSize:(l
ong long)uploadedSize totalSize:(long long)totalSize  {
}
//上传token过期
- (void)publishManagerUploadTokenExpired:(AliyunVodPublishManager *)manager {
}
//上传超时，开始重试
- (void)publishManagerUploadRetry:(AliyunVodPublishManager *)manager {
}
//重试结束，继续上传
- (void)publishManagerUploadRetryResume:(AliyunVodPublishManager *)manager {
}
```

上传控制

上传过程中可按需暂停、继续以及取消上传。

```
//暂停上传
[AliyunVodPublishManager pauseUpload];
//继续上传
[AliyunVodPublishManager resumeUpload];
//取消上传
[AliyunVodPublishManager cancelUpload];
```

5.10. 视频工具

iOS短视频SDK提供了视频文件解析、视频缩略图获取工具以辅助用户更方便的编辑录制视频等。

版本支持

版本	是否支持
专业版	支持
标准版	支持
基础版	支持

相关类功能

类名	功能
AliyunNativeParser	工具解析核心类，解析视频、音频及文件相关信息。
AliyunThumbnailParser	缩略图核心类，设置缩率图的裁剪范围、输出大小、开始取消等参数。

视频文件解析

视频文件解析的核心类为AliyunNativeParser类，通过该类可以解析出如下信息：

- 视频：视频流编码格式、视频宽高、开始时间、时长、码率、帧率、关键帧间隔、总帧数、格式、旋转角度等信息。
- 音频：音频流编码格式、开始时间、时长、总帧数、采样率、音频轨道数、码率、音频格式等信息。
- 文件：文件名称、开始时间、时长、文件格式等信息。

```
//1.初始化
AliyunNativeParser *parser = [[AliyunNativeParser alloc] initWithPath:videoPath];

//2.获取视频信息
//通过接口获取
NSInteger videoWidth = [parser getVideoWidth];
NSInteger videoHeight = [parser getVideoHeight];

//通过Key的方式获取
float videoDuration = [parser getValueForKey:ALIYUN_VIDEO_DURATION].floatValue;
NSInteger videoBitrate = [parser getValueForKey:ALIYUN_VIDEO_BIT_RATE].integerValue;
```

视频缩略图

既可通过视频缩略图的核心类AliyunThumbnailParser类获取缩略图，也可通过AliyunNativeParser类获取缩略图，通过这些类可以在视频轨道上展示每一帧图片的缩略图，通常在视频编辑或裁剪时需要用到。

- 使用AliyunNativeParser类获取

```
//1.初始化
self.parser = [[AliyunNativeParser alloc] initWithPath:videoPath];

//2.获取缩略图
//使用时间间隔获取接口
[self.parser loadThumbnailListWithDuration:1.0 imageWidth:100 complete:^(int errorCode, N
SArray<UIImage *> *imageList) {
 }];

//使用时间数组获取接口
NSArray<NSNumber *> *timeList = @[@0,@1.0,@2.5,@6];
 [self.parser loadThumbnailWithTimeList: timeList imageWidth:100 complete:^(int errorCode
, NSArray<UIImage *> *imageList) {
 }];
```

> **说明**
>
> 获取缩略图是一个异步操作，过程中请保证AliyunNativeParser实例不被释放。

- 使用AliyunThumbnailParser类获取

```
//1.初始化
self.parser = [[AliyunThumbnailParser alloc]initWithPath:self.videoPath delegate:self];

//2.设置图片参数
//图片裁剪范围
[self.parser setCutFrame:CGRectMake(0, 0, 200, 200)];

//图片输出大小
CGFloat ouputSizeWidth = 200;
[self.parser setOutputSize:CGSizeMake(ouputSizeWidth, ouputSizeWidth)];

//3.设置缩略图的时间点
NSArray<NSNumber *> *timeList = @[@0,@1.0,@2.5,@6];
[self.parser addThumbnailTimeList:timeList];

//4.开始获取缩略图
[self.parser start];

//5.获取缩略图回调
// 获取出错，注意在子线程回调
- (void)thumbnailParser:(AliyunThumbnailParser *)parser onError:(int)code {
}

//获取某张图片出错，注意在子线程回调
- (void)thumbnailParser:(AliyunThumbnailParser *)parser onPicError:(int)code time:(float)ti
me {
}

//获取到的图片，注意在子线程回调
- (void)thumbnailParser:(AliyunThumbnailParser *)parser onGetPicture:(UIImage *)image time:
(float)time {
}

//获取图片完成，注意在子线程回调
- (void)thumbnailParserOnCompleted:(AliyunThumbnailParser *)parser {
}
```

6.资源和特效

6.1. 动图

短视频SDK中的动图资源支持自定义，自定义的动图需满足一定的制作规范才能够正常使用。

制作规范

自定义动图制作规范，请参见阿里云短视频SDK动态贴纸及气泡制作规范。

调用方法

- 动态贴纸
  - Android

    调用 `AliyunPasterManager.addPasterWithStartTime(Source path, long startTime, long duration);` 方法添加动态贴纸特效。接口参数请参考AliyunPasterManager。
  - iOS

    调用 `AliyunStickerManager` 的 `addGif:startTime:duration:` 添加动态贴纸特效，接口参数请参考AliyunStickerManager。
- 气泡文字
  - Android

    调用 `AliyunPasterControllerCompoundCaption.setBubbleEffectTemplate(Source aStyleTemplate);` 方法添加气泡文字特效。接口参数请参考AliyunPasterControllerCompoundCaption。
  - iOS

    调用 `AliyunStickerManager` 的 `addCaptionText:bubblePath:startTime:duration:` 添加气泡文字特效。接口参数请参考AliyunStickerManager。

6.2. 滤镜及转场

短视频SDK提供了自定义特效能力，您可以基于通用特效配置文件，通过自定义OpenGL ES Shader（OpenGL ES 3.0语法），实现想要的滤镜与转场效果。有关OpenGL ES的知识点和解释并不属于本篇文档的范畴，我们默认您已经对OpenGL ES及其Shader Language有所了解。

简介

一个滤镜或转场的特效资源包文件夹，包含了一个名为config.json的通用特效配置文件以及一些图片素材。在短视频SDK中编辑视频和录制视频时，支持使用转场和滤镜特效，即调用相关编辑和录制接口时，传入所配置的特效资源包文件夹目录，来应用一个特效效果。

配置通用特效配置文件

通用特效配置文件采用JSON格式，描述了完整的渲染过程。整体结构分为两个层级，第一层级描述了特效的基本信息，第二层级用节点树描述了特效的实现细节。

特效基本信息-第一层级

特效的基本信息包含以下字段：

字段	说明
name	特效名称。
module	模块标识，该字段必须是 `ALIVC_GECF` 。
version	版本号，当前版本为1。
type	特效类型，1表示滤镜，2表示转场。
nodeTree	节点树，用于描述特效的实现细节，详细内容请参见节点树-第二层级。

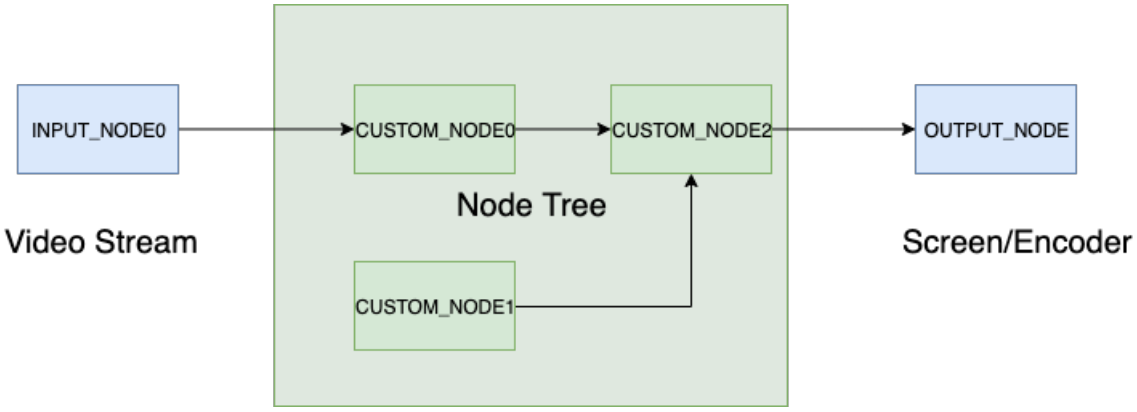
以特效Intense滤镜的配置文件为例，示例如下：

```
{
    "name": "Intense",
    "module": "ALIVC_GECF",
    "version": 1,
    "type": 1,
    "nodeTree": [
        {
            "nodeId": 0,
            "name": "Intense",
            "fragment": "/** ..... */",
            "textures": [
                {
                    "name": "inputImageTexture",
                    "srcType": "INPUT_NODE",
                    "nodeId": 0
                },
                {
                    "name": "inputImageTexture2",
                    "srcType": "IMAGE",
                    "path": "color.png"
                }
            ]
        }
    ]
}
```

节点树-第二层级

节点树 `nodeTree` 用于描述一个渲染流程，包含了一个或多个节点。

在短视频SDK中，渲染流程如图所示：



据上图所示，渲染过程被抽象为一系列节点组成的树状结构。

- 输入节点INPUT_NODE代表原始输入源。在录制场景下，输入节点是摄像头采集的图像数据流。在编辑场景下，输入节点是当前播放视频流。在设计上，输入节点可以是多个。例如转场过程中，需要对前后两个视频做效果变换。此时，前一个视频是INPUT_NODE0，后一个是INPUT_NODE1。
- 中间的节点树部分即对应配置文件中的 `nodeTree` 字段。一个节点树可以包含一个或多个渲染节点。可以看到，整个渲染配置文件的关键就是有关节点的配置。
- 输出节点OUTPUT_NODE代表渲染后的视频流。在预览模式下，输出节点输出到屏幕。在合成模式下，输出节点输出到编码器编码。

节点

节点字段描述了整个渲染流程中，某一次绘制过程中的相关配置，这里的配置包含了自定义特效所必须的着色器代码以及相关参数描述。 节点包含以下字段：

字段	说明
nodeId	节点id，用于标识当前节点。
name	节点名称。

字段	说明

vertex	顶点着色器代码。
	该字段作用与vertexPath字段相同，声明其中任意一个即可。
	该字段可以不写，如果不填写，SDK会提供默认实现如下：
	`attribute vec4 position;`
	`attribute vec4 inputTextureCoordinate;`
	`varying vec2 textureCoordinate;`
	`void main()`
	`{`
	`gl_Position = position;`
	`textureCoordinate = inputTextureCoordinate.xy;`
	`}`

vertexPath	顶点着色器代码所在文件路径。 该字段作用与vertex字段相同，声明其中任意一个即可。

attributes	attributes列表，该字段用于声明顶点着色中attribute参数名name和类型type。
	type取值为：POSITION，顶点坐标；TEXTURECOORD，纹理坐标。
	该字段可以不填写，如果不填写，SDK会提供默认实现如下：
	`[`
	`{`
	`"name": "position",`
	`"type": "POSITION"`
	`},`
	`{`
	`"name": "inputTextureCoordinate",`
	`"type": "TEXTURECOORD"`
	`}`
	`]`

fragment	片段着色器代码。 该字段作用与fragmentPath字段相同，声明其中任意一个即可。

字段	说明
fragmentPath	片段着色器代码所在文件路径。 该字段作用与fragment字段相同，声明其中任意一个即可。

textures	纹理列表。用于描述片段着色器中 `sampler2D` 纹理参数的相关属性。属性包括：
	• name：纹理名。
	• srcType：纹理数据源类型，代表待绘制的纹理数据从哪里获取。支持类型如下：
	  ○ IMAGE：当前纹理来自资源图片，该类型需要同时声明 `path` 字段，并且把图片文件放到资源包中
	  ○ INPUT_NODE：当前纹理来自输入节点传入的图像数据，该类型需要同时声明nodeId字段
	  ○ CUSTOM_NODE：当前纹理来自自定义节点处理后的图像数据，该类型需要同时声明nodeId字段。
	• nodeId：节点id。
	  ○ srcType为CUSTOM_NODE或INPUT_NODE时需要填写。
	  ○ 当srcType为INPUT_NODE时，nodeId字段不能随意填写。滤镜场景下为0。转场场景下前一个视频流nodeId为0，后一个视频流nodeId为1。
	• path：资源文件路径，srcType为IMAGE时需要填写。

params	参数列表。用于描述片段着色器中 `uniform` 参数的相关属性。属性包括： • name：参数名。 • type：参数类型。支持的类型请参见params参数类型（type）支持的类型。 • value：参数取值。 • maxValue：参数最大值。 • minValue：参数最小值。

params参数类型（type） 支持的类型

类型	参数取值示例
INT	[1]
FLOAT	[-2.0]
VEC2I	[3, 2]
VEC3I	[1, 2, 3]
VEC4I	[4, 3, 2, 1]
VEC2F	[-1.0, 1.0]
VEC3F	[0.5, 0.5, 0.5]
VEC4F	[1.0, -1.0, 1.0, -1.0]
MAT3F	[1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
MAT4F	[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

以特效**Translate转场**的配置文件为例，示例如下：

```
{
    "name": "Translate",
    "module": "ALIVC_GECF",
    "version": 1,
    "type": 2,
    "nodeTree": [
        {
            "nodeId": 0,
            "name": "Translate",
            "vertex": "/** ..... */",
            "attributes": [
                {
                    "name": "a_position",
                    "type": "POSITION"
                },
                {
                    "name": "a_texcoord",
                    "type": "TEXTURECOORD"
                }
            ],
            "fragment": "/** ..... */",
            "textures": [
                {
                    "name": "RACE_Tex0",
                    "srcType": "INPUT_NODE",
                    "nodeId": 0
                },
                {
                    "name": "RACE_Tex1",
                    "srcType": "INPUT_NODE",
                    "nodeId": 1
                }
            ],
            "params": [
                {
                    "name": "direction",
                    "type": "INT",
                    "value": [
                        0
                    ],
                    "maxValue": [
                        3
                    ],
                    "minValue": [
                        0
                    ]
                }
            ]
        }
    ]
}
```

内建变量

SDK中内置了一些变量，您可以在shader中直接声明，目前有以下内建变量：

```
uniform highp float BUILTIN_PROGRESS; // 转场进度：0~1
uniform highp float BUILTIN_WIDTH;    // 图像宽
uniform highp float BUILTIN_HEIGHT;   // 图像高
```

在短视频SDK中使用特效

使用特效通常需要先创建特效对象，再应用特效，最后按需更新特效参数。

初始化后的特效对象内可以获取到描述特效配置的 `AliyunEffectConfig` 对象，该对象内部结构与特效配置文件的结构相对应。 如果某一个自定义特效配置文件里包含有 `params字段` ，对应在代码中可以通过 `AliyunEffectConfig` -> `nodeTree` -> `params` 获取到 `AliyunParam` 对象。 `AliyunParam` 对象中的 `value` 字段就是当前参数的值。更新特效参数需要以下两步：

1. 通过 `AliyunValue` 对象提供的update设值方法更新参数值。
2. 调用特效更新方法更新参数。具体特效更新方法请参见如下操作步骤中的第三步更新特效参数。

Android操作步骤

- 滤镜
  i. 调用 `EffectFilter(String path)` 方法创建特效对象， `path` 参数为特效文件夹路径。
  ii. 调用 `int addAnimationFilter(EffectFilter filter);` 方法应用滤镜特效。
  iii. 调用 `int updateAnimationFilter(EffectFilter filter);` 方法更新滤镜特效参数。
- 转场
  i. 调用 `TransitionBase(String path)` 创建特效对象， `path` 参数为特效文件夹路径。
  ii. 调用 `int setTransition(int index, TransitionBase transition);` 方法应用转场特效。
  iii. 调用 `int updateTransition(int clipIndex, TransitionBase transitionBase);` 方法更新转场特效参数。

iOS操作步骤

- 滤镜
  i. 调用 `AliyunEffectFilter` 的 `- (instancetype)initWithFile:(NSString *)path;` 方法创建特效对象， `path` 参数为特效文件夹路径。
  ii. 调用 `- (int)applyAnimationFilter:(AliyunEffectFilter *)filter;` 方法应用滤镜特效。
  iii. 调用 `- (int)updateAnimationFilter:(AliyunEffectFilter *)filter;` 方法更新滤镜特效参数。

- 转场

  i. 调用 `AliyunTransitionEffect` 的 `-(instancetype)initWithPath:(NSString *)path;` 方法创建特效对象，`path` 参数为特效文件夹路径。

  ii. 调用 `- (int)applyTransition:(AliyunTransitionEffect *)transition atIndex:(int)clipIdx;` 方法应用转场特效。

  iii. 调用 `-(int)updateTransition:(AliyunTransitionEffect *)transition atIndex:(int)clipIdx;` 方法更新转场特效参数。

6.3. 花字

短视频SDK提供了丰富的功能接口，产品级的交互，丰富的视频素材库（MV、贴纸、气泡、花字），并推出了特效自定义能力来满足不同场景和行业下的素材需求，本文档将重点介绍花字的制作规范和输出。

简介

花字是基于字幕的一种特效，一个花字特效资源包文件夹，包含了一个名为config.json的通用特效配置文件以及一些图片素材。您可以通过短视频SDK提供的编辑接口，传入特效资源包文件夹目录，来应用一个花字特效效果。

一个完整的花字特效资源包通常包含以下三个部分：

- config.json：必选，花字的配置文件。配置说明参见config.json配置说明。
- icon.png：可选，花字缩略图，即最终生成的花字效果缩略图。
- lieheng.png：可选，花字文字贴图，即基于此图片设置花字效果，当配置文件有配置时使用。

配置花字特效资源包

config.json配置说明如下表所示：

config.json配置说明

字段	类型	是否必选	说明
version	String	是	配置版本号，默认为1。
color	String	是	字体颜色，格式：#AARRGGBB或#RRGGBB。
texture	String	否	花字文字贴图，只支持jpg和png格式，且需和config.json配置文件放置在同一文件夹下。目前花字文字贴图的规则是每个文字的贴图完全一样，取贴图覆盖到字体颜色区域部分。
outline1	JSONObject	是	第一层描边。属性说明参见outline属性说明。

字段	类型	是否必选	说明
outline2	JSONObject	否	第二层描边，和outline1数据结构相同。属性说明参见outline属性说明。

outline包含outline1和outline2，outline属性说明如下表所示：

outline属性说明

字段	类型	是否必选	说明
type	String	是	默认为normal，代表默认使用系统字体。
data	JSONArray	是	定义了描边的一系列颜色及宽度，详细请参见data子节点属性说明。当type为normal时，data最多允许有3个颜色配置，超过3个，则只会取前3个颜色配置。

data子节点属性说明如下表所示：

data子节点属性说明

字段	类型	是否必选	说明
color	String	是	描边颜色，格式：#AARRGGBB或#RRGGBB。
width	float	是	描边宽度，取值范围：[0~64]，且下一个同类型节点的width值，必须大于当前width值。说明 每一个outline的data子节点的width属性值，必须大于上一个data子节点的width属性值，示例如下。

正确的width取值样例如下，data5的width > data4的width > data3的width > data2的width > data1的width。

```
{
  "outline1": {
    "type": "normal",
    "data":[   // data1
      {
        "color": "#5350DD",
        "width": 2
      },
      {    // data2
        "color": "#B5FAA7",
        "width": 4
      }
    ]
  },
  "outline2": {
    "type": "normal",
    "data":[
      {    // data3
        "color": "#6E58F8",
        "width": 8
      },
      {    // data4
        "color": "#69F88C",
        "width": 10
      },
      {    // data5
        "color": "#FA55D8",
        "width": 12
      }
    ]
  }
}
```

配置示例

本文以设置如下花字效果为例进行示例说明。该示例的配置资源包Demo：示例Demo.zip。

- 该示例Demo包的文件夹结构如下：

  ├── config.json

  ├── icon.png

  └── lieheng.png

- config.json的配置如下：

```
{
  "version": 1,
  "color":"#000000",
  "texture": "lieheng.png",
  "outline1": {
    "type": "normal",
    "data":[
      {
        "color": "#ffffff",
        "width": 8
      }
    ]
  },
  "outline2": {
    "type": "normal",
    "data":[
      {
        "color": "#000000",
        "width": 15
      }
    ]
  }
}
```

在短视频SDK中使用特效

目前仅支持在编辑视频时，使用花字特效。

- Android

  调用 `AliyunPasterControllerCompoundCaption.setFontEffectTemplate` 方法应用花字效果。详细操作请参考设置字幕及动态贴纸。

- iOS

  调用 `AliyunCaptionStickerController.setFontEffectTemplate` 方法应用花字效果。详细操作请参考设置字幕及贴纸。

6.4. MV

短视频SDK中的MV资源支持自定义，自定义的MV需满足一定的制作规范才能够正常使用。本文为您介绍自定义MV的制作规范及第三方资源。

制作规范

自定义MV制作规范，请参见阿里云短视频SDK MV制作规范。

第三方资源

中旌影视提供了海量的短视频创意主题模板（MV），拥有强大的后期制作，短视频特效素材创作能力。特效制作有第三方费用，可直接和合作伙伴沟通。

使用说明

在编辑视频时可添加MV，详细信息请参见编辑视频（Android）和编辑视频（iOS）。

6.5. 人脸AR

短视频SDK提供的高级美颜和美肌功能由阿里云美颜特效SDK和相芯科技（FaceUnity）提供服务。

美颜特效SDK

简介

接入简单快速、性能高、功能多样，是阿里云美颜特效SDK的核心优势。美颜特效SDK为视频创作者提供移动端和PC端的人脸基础美颜、高级美颜、美型美妆、贴纸抠像、手势姿态识别等编辑加工能力，基于自研的智能视觉算法、海量规模的人脸、人体检测和识别技术，为满足直播和视频制作时的美颜特效需求提供各种支持。

使用说明

想要在短视频SDK中使用美颜特效SDK所提供的特效，则需要提前获取美颜特效SDK权限并将美颜特效SDK集成接入到短视频SDK中。

有关美颜特效SDK的权限获取、集成接入、特效使用等相关操作请参见美颜特效SDK。

FaceUnity

简介

FaceUnity提供了短视频场景中视频的美颜（磨皮、美白、红润等）、美肌（大眼瘦脸）等能力。有第三方费用，可通过云市场和合作伙伴沟通。

使用说明

有关FaceUnity的购买、集成、使用等详情，请参见人脸AR方案。

6.6. 在线音乐

短视频SDK支持将网络音乐和本地音乐合成到视频中，支持任意时间点添加。在线音乐资源由第三方提供，需额外付费。

在线音乐资源

短视频SDK示例代码中提供的示例音乐由太合音乐DMH提供服务。DMH服务涵盖不同曲风、万首高质量可转授歌曲，提供了SDK、API等对接方式。在线音乐有第三方费用，按调用次数计费，可直接和合作伙伴沟通。详情请参见在线音乐。

7.短视频SDK参考文档

7.1. 错误码

本文按不同功能介绍短视频SDK的错误码。

通用

错误码	说明
0	成功。
-1	失败。
-2	参数错误。
-3	未知错误。
-4	状态错误。

视频封装器（Video Muxer）

错误码	说明
-10001001	创建封装上下文失败。
-10001002	创建封装音频上下文失败。
-10001003	创建封装流失败。
-10001004	创建封装视频上下文失败。
-10001005	写入封装数据包失败。
-10001006	设置音视频流参数异常。
-10001007	封装服务状态异常。

错误码	说明
-10001008	写入封装文件头失败。
-10001009	写入封装文件尾失败。
-10001010	函数传入参数错误，如传入空的输入路径名、输出路径名、音频采样率、音频通道数、分辨率等。
-10001011	无法找到封装指定的编码器类型。
-10001012	创建封装流失败。
-10001013	创建封装上下文失败。
-10001014	创建封装IO上下文失败。

视频解封装器（Video Demuxer）

错误码	说明
-10002001	文件解析失败。
-10002002	文件重复解析。
-10002003	文件打开失败。
-10002004	文件流未找到。
-10002005	文件解析结束出错。

视频编码器（Video Encoder）

错误码	说明
-10003001	创建视频编码器失败，参数不支持。

错误码	说明
-10003002	创建视频编码器失败，没有符合该编码场景要求的编码器。
-10003003	视频编码组件没有设置输出数据的接收方，无效工作。
-10003004	视频编码器被打断，运行中报错。
-10003005	系统版本限制导致视频硬编码器创建失败。
-10003006	视频编码器启动失败。
-10003007	视频帧输入数据不合法。
-10003008	视频编码器内部缓存已满，需要等待。
-10003009	iOS视频硬编码session报错。
-10003010	输入数据的pts非法，不符合编码要求。
-10003011	视频编码器被暂停，等待唤醒。
-10003012	视频编码器被暂停，等待唤醒。
-10003013	视频编码器送空，解码失败。
-10003101	OpenH264初始化失败。
-10003102	OpenH264输入数据不合法。
-10003103	OpenH264初始化编码器失败。
-10003104	OpenH264格式不支持。
-10003105	OpenH264编码失败。

错误码	说明
-10003106	OpenH264编码无输出。
-10003107	OpenH264编码SPS解析失败。
-10003201	MediaCodec状态错误。
-10003202	MediaCodec Buffer错误。
-10003203	MediaCodec其他错误。
-10003204	Android编码输入NULL。
-10003205	Android编码output句柄为NULL。
-10003206	Android编码feat不支持。
-10003207	Android编码format不支持。
-10003208	Android编码初始化失败。
-10003209	Android编码获取输出帧失败。
-10003210	Android编码更新码率失败：码率超出有效范围。
-10003211	Android编码更新码率失败：编码器已释放。
-10003212	Android编码更新码率失败：其他。
-10003213	Android编码更新码率失败：视频宽高超过4096。
-10003301	FFmpeg创建失败。

错误码	说明
-10003302	FFmpeg找不到编码器。
-10003303	FFmpeg编码失败。
-10003304	FFmpeg编码失败。
-10003401	混合编码初始化失败。
-10003402	混合编码初始化硬编失败。
-10003403	混合编码无法启动解码。
-10003404	混合编码初始化软编失败。
-10003405	混合编码无法启动软编服务。

视频解码器（Video Decoder）

错误码	说明
-10004001	无SPS。
-10004002	创建H264解码失败。
-10004003	创建HEVC解码失败。
-10004004	创建解码失败。
-10004005	解码状态错误。
-10004006	创建解码失败。
-10004007	解码没有足够缓存。

错误码	说明
-10004008	解码被中断。
-10004009	解码SPS失败。
-10004010	解码资源失败。
-10004011	解码失败。
-10004101	软解不支持设置OES格式输出。
-10004102	软解找不到codec对应的解码器。
-10004103	软解解码找不到上下文。
-10004104	软解解码创建上下文失败。
-10004105	软解内部打开失败。
-10004106	软解解码一帧失败。
-10004107	软解未知的输出格式。
-10004108	软解未知的输入帧类型。
-10004109	软解Flush失败。
-10004201	Android硬解不支持，因为命中黑名单。
-10004202	Android硬解不支持，因为编码类型。
-10004203	Android硬解创建SurfaceTexture失败。
-10004204	Android硬解mediacodec初始化失败。

错误码	说明
-10004205	Android硬解mediacodec configure失败。
-10004206	Android硬解未知的输入帧类型。
-10004207	Android硬解mediacodec queueInput失败。
-10004208	Android硬解mediacodec start失败。
-10004209	Android硬解mediacodec dequeueInput失败。
-10004210	Android硬解mediacodec dequeueInput失败。
-10004211	Android硬解mediacodec dequeueOut失败。
-10004212	Android硬解mediacodec queueIn失败。
-10004213	Android硬解mediacodec queueIn失败。
-10004214	Android硬解mediacodec queueIn失败。
-10004215	Android硬解mediacodec flush失败。
-10004216	Android硬解flush超时。
-10004217	Android硬解flush中断。
-10004302	iOS硬解码：版本太低导致hevc失败。
-10004303	iOS硬解码：硬件不支持。
-10004304	iOS硬解码：不支持解码类型。
-10004305	iOS硬解码：输入帧有误。

错误码	说明
-10004306	iOS硬解码：输入帧解析失败。
-10004308	iOS硬解码：编译平台错误。
-10004309	iOS硬解码：createSession失败。
-10004310	iOS硬解码：createBuffer失败。
-10004311	iOS硬解码：createSampleBuffer失败。
-10004312	iOS硬解码：解码失败kVTInvalidSessionErr。
-10004313	iOS硬解码：解码失败。
-10004314	iOS硬解码：解码回调错误。
-10004315	iOS硬解码：创建format失败。
-10004316	iOS硬解码：无video format。

音频编码器（Audio Encoder）

错误码	说明
-10005001	创建音频编码器失败，参数不支持。
-10005002	音频编码组件状态不对。
-10005003	音频帧输入数据不合法。
-10005004	音频编码器被打断，无法继续工作。
-10005005	音频编码组件没有设置数据输出的接收端，无意义工作。

错误码	说明
-10005006	没有符合编码场景要求的编码器。
-10005007	open失败。
-10005008	set bitrate失败。
-10005009	set sample rate失败。
-10005010	set aot失败。
-10005011	set transmux失败。
-10005012	set singlemode失败。
-10005013	set channel失败。
-10005014	set channelloader失败。
-10005015	set afterburner失败。
-10005016	get info失败。
-10005017	没有编码器。
-10005018	音频初始化输入config不合法。
-10005019	重复初始化。
-10005020	创建编码器失败。
-10005021	没有符合编码场景要求的编码器。
-10005022	没有编码器实例。

音频解码器（Audio Decoder）

错误码	说明
-10006001	创建音频解码器失败，参数不支持。
-10006002	音频解码组件状态不符，出现了重复创建或者重复销毁。
-10006003	输入的音频packet数据不合法。
-10006004	音频解码器内部缓存已满，需要等待。
-10006005	音频解码器内部打开失败。
-10006006	不支持的音频解码类型。
-10006007	音频解码器解码一帧失败。
-10006008	音频解码器创建无内存。
-10006009	解码器已经存在，不能再次创建。
-10006010	创建音频解码器失败，参数不支持。
-10006011	解码器已经存在，不能再次创建。
-10006012	解码器已经存在，不能再次创建。
-10006013	解码器创建失败。
-10006014	解码器输入不正确。
-10006015	解码器cookiedata设置失败

渲染

错误码	说明
-10007001	第一帧已渲染。
-10007002	EGL报错。
-10007003	GL报错。
-10007004	待渲染的数据非法。
-10007006	渲染的窗口大小非法。
-10007007	渲染节点编排异常。
-10007008	渲染节点参数设置异常。
-10007009	渲染编排构建异常。
-10007010	帧动画传入参数异常。
-10007011	GL上下文创建失败。
-10007013	eglSwapBuffers失败。

数据池

错误码	说明
-10008001	数据池状态错误。
-10008002	数据池处理失败。
-10008003	磁盘空间不足。
-10008004	数据池解析视频GOP出错。

错误码	说明
-10008005	数据池音频流初始化失败。
-10008006	数据池视频流初始化失败。
-10008007	数据池缓冲区溢出。
-10008008	数据池媒体流不存在。

音频处理

错误码	说明
-10009001	创建失败。
-10009002	重新创建。
-10009003	销毁失败。
-10009004	重复销毁。
-10009005	流配置失败。
-10009006	属性配置失败。
-10009007	送算法模块失败。
-10009008	内存满。
-10009009	配置应用失败。
-10009010	拖动失败。
-10009011	未知配置。

错误码	说明
-10009012	Push空数据。
-10010001	内核创建失败。
-10010002	音频渲染重复创建。
-10010003	设置静音失败。

License

错误码	说明
-10011001	License无效，可能原因：过期或者未授权。
-10011002	License校验异常，无MV权限。
-10011003	License校验异常，无动图或字幕权限。
-10011004	License校验异常，无裁剪权限。
-20001001	License无效。

短视频

错误码	说明
-20001002	未支付该特效功能。
-20001003	特效使用资源不存在。
-20001004	特效不能覆盖已有的效果。
-20001005	特效使用失败。

错误码	说明
-20001006	没有设备权限（音频采集、文件读写、摄像头使用、网络访问等Android设备权限）。
-20001007	设置背景音乐时参数错误，像背景音乐路径及时间参数。
-20001008	设置背景音乐时错误，音频格式不支持。

TRES

错误码	说明
-20002001	播放器未准备好。
-20002002	多次调用异常。
-20002003	音频输入流异常。
-20002004	视频渲染类异常。
-20002005	重复添加特效。
-20002006	不合法的状态。

参数相关

错误码	说明
-20003001	参数异常。
-20003002	无效参数。
-20003003	输入路径为空。
-20003004	输入视频路径为空。

错误码	说明
-20003005	文件不存在。
-20003006	片段索引无效，超出clip列表最大值或者索引不合法。
-20003007	片段无效，片段为null。
-20003008	当前列表已经没有片段了，不能执行删除操作。
-20003009	视频尺寸设置不正确。
-20003010	视频或音频时长设置异常。
-20003011	动图路径异常。
-20003012	资源解析异常。
-20003013	图片路径异常。
-20003014	滤镜路径异常。
-20003015	PictureSize不合法。
-20003016	水印对象为空。
-20003017	无效的涂鸦。
-20003018	特效作用时间区间冲突（两个动效滤镜时间区间有交叉）。
-20003019	project为空。
-20003020	添加媒体文件失败。
-20003021	输出路径无效。

错误码	说明
-20003022	MV路径异常。

Media相关

错误码	说明
-20004001	不支持的视频格式。
-20004002	不支持的音频格式。
-20004003	不支持的图片格式。
-20004004	不支持的媒体类型。
-20004005	视频编码器内部错误。
-20004006	音频编码器内部错误。
-20004007	裁剪图片失败。
-20004008	视频解码器内部错误。
-20004009	视频编码器参数错误。
-20004010	音频编码器参数错误。
-20004011	音频解码器内部错误。
-20004012	不支持的像素格式。
-20004013	播放器内部错误。
-20004014	不支持的像素格式。

错误码	说明
-20004015	编码组件初始化失败。
-20004016	解析输入文件失败。
-20004017	视频队列为空。
-20004018	音频队列为空。
-20004019	音频处理最大仅支持两路输入。
-20004020	音频处理初始化失败。
-20004021	MV配置文件解析失败。

IO相关

错误码	说明
-20005001	开启预览失败。
-20005002	开启摄像头失败。
-20005003	切换摄像头失败。
-20005004	创建临时录制文件失败。
-20005005	音频录制失败。
-20005006	摄像头设置参数错误。
-20005007	摄像头未开启。
-20005008	不支持的摄像头类型。

视频录制

错误码	说明
-20008001	视频输入句柄设置为空。
-20008002	视频输入句柄设置为空。
-20008003	音频输入句柄设置为空。
-20008004	结束录制失败。
-20008005	停止录制未完成。
-20008006	录制编码器初始化失败。
-20008007	录制未准备好。
-20008008	录制已达到最大时长。
-20008009	录制Service创建失败。
-20008010	录制Proxy创建失败。
-20008011	录制Service状态错误。
-20008012	录制采集的数据不支持。
-20008013	SurfaceTexture类信息找不到。
-20009001	合拍渲染布局失败。

视频编辑

错误码	说明
-20011001	编辑模块系统性错误，如编解码错误。
-20011002	编辑准备阶段出错。
-20011003	编辑开始阶段出错（开始播放，开始合成）。
-20011004	编辑停止阶段出错（停止播放，停止合成）。
-20011005	编辑暂停阶段出错（暂停播放，暂停合成）。
-20011006	编辑继续阶段出猝（继续播放，继续合成）。
-20011007	编辑快进出错。
-20011008	编辑时间特效不支持多段视频，包括时间特效反复，倒播。
-20011009	编辑初始化失败。
-20011010	编辑切换模式，反初始化前一种模式的组件失败。
-20011011	编辑设置模式时没有找到对应的显示输出设置。
-20011012	快进间隔时间太小。
-20011013	编辑设置显示参数失败。
-20011014	编辑时找不到对应的文件。
-20011015	特效资源解析失败。
-20011016	不支持的图片特效类型。
-20011017	Demuxer解析文件失败。

错误码	说明
-20011018	合成状态出错。
-20011019	编辑状态错误。
-20011020	无效的合成,合成需要的参数字符串为空。
-20011021	合成初始化失败。
-20011022	无效的转场时长。
-20011023	合成需要的组件为空，一般是状态不对。
-20011024	编辑未初始化。

视频上传

错误码	说明
-20012001	上传失败。
-20012002	上传参数为空。
-20012003	上传状态错误。

其他错误

错误码	说明
-20006001	录制失败，来电占用。
-20006002	退后台报错。
-20007001	缩略图内部逻辑错误。

错误码	说明
-20007002	缩略图色彩空间转换初始化失败。
-20007003	缩略图色彩空间转换失败。
-20007004	缩略图解码器内部错误。
-20007005	缩略图流解析错误。
-20007007	缩略图准备失败。
-20010001	转码状态错误。
-20010002	错误的转码参数。
-20010003	转码退后台错误。
-20010004	转码初始化参数无效。

7.2. API参考

本文提供短视频SDK客户端接口地址。

Android端接口

Android端接口文档以V3.17.0版本为界有所不同，且接口区分语言。详情如下：

版本	接口地址	说明
V3.17.0及以上	• 中文接口：接口文档 • 英文接口：API Reference	不涉及。
V3.17.0以下	• 中文接口：接口文档 • 英文接口：API Reference	旧接口转换新接口需要辅助工具。如需转换，请下载辅助工具。

iOS端接口

iOS端接口地址区分语言，不区分版本。

- 中文版接口文档地址：接口文档
- 英文版接口文档地址：API Reference

7.3. GitHub文档

短视频SDK文档已在GitHub开源，方便开发者使用。

文档地址请参见短视频SDK GitHub文档。

8.短视频SDK常见问题

8.1. 短视频SDK和License常见问题

本文主要介绍使用短视频SDK时遇到的SDK使用、试用、购买及License等问题。

如何申请试用短视频SDK?

请发送以下信息至指定邮箱videosdk@service.aliyun.com。收到申请邮件后，我们将于2个工作日之内处理完开通申请。

- 公司名称
- 应用名称
- 申请试用的SDK版本（基础、标准、专业）
- 联系人
- 联系电话
- 应用bundleID
- 包名和签名信息（MD5）
  - 要求：MD5格式、小写、无冒号。
  - 可使用签名获取工具来获取包名和签名信息，安装后按工具中的提示操作。
- 阿里云账号或UID

  如没有阿里云账号，请提前注册。注册步骤请参见注册阿里云账号。

> 注意
>
> - License免费试用期为一个月。
> - 请确保申请信息齐全、格式规范，后续若有需要更换包名、签名文件、BundleID任意一项，请重新申请。
> - 短视频专业版SDK的试用版支持高级美颜、智能抠图和手势识别等功能，这些功能由阿里云简介或第三方SDK提供。如需试用，需要在申请邮件中额外注明，美颜特效SDK或第三方SDK的授权证书会通过邮件的形式送达。

如何免费使用短视频SDK?

目前购买点播套餐即可免费使用短视频SDK，详细请参见获取短视频SDK License。

如何购买专业版短视频SDK?

发送试用邮件之后会有商务经理同您沟通。具体专业版和其他版本的区别，请参见：功能列表。

提示License Failed，该怎么办?

1. 确保已购买或获赠或申请试用的短视频SDK License未到期，并已发送邮件至邮箱videosdk@service.aliyun.com开通短视频SDK License授权。有关查询Licsense是否到期的方法，请参见获取短视频SDK License。
2. 确认申请试用的信息是否正确。
   - 如果是iOS端，请确保应用bundleID和申请开通的应用bundleID是一致的。

- 如果是Android端，请确保包名和签名信息（MD5）获取是否一致。使用签名获取工具，安装后按工具中的指导操作，来获取正确的包名和签名信息。

人脸贴纸与人脸AR、动态贴纸的区别？

人脸贴纸属于视频录制中的功能，即通过人脸识别的功能，给识别到的人脸添加指定的贴纸，目前仅专业版本支持。 人脸AR属于美颜特效SDK中的功能，需要额外授权。 动态贴纸属于视频编辑中的功能，仅专业版本支持。

8.2. Android端短视频SDK常见问题

本文主要介绍短视频SDK在Android端的使用问题。

SDK集成

安卓指令集的兼容情况如何？

如果项目只接入armeabi，则需要将SDK包中的armeabi_v7a文件内的so文件拷贝到项目的armeabi文件夹内。

集成SDK后，Debug版本可以正常运行，但是Release版本启动就崩溃，该怎么解决？

问题现象：集成SDK后，Debug版本可以正常运行，但是Release版本启动就崩溃。

可能原因：混淆配置错误

解决方案：

1. 先检查崩溃的log是否是报JNI找不到对应的Java类。
   - 如果是的话，一般来说就是混淆导致的，因为JNI调用Java类用的是反射，所以如果混淆把SDK内部与JNI有关的类混淆了，则JNI加载时将无法找到对应的Java类，就会加载失败。
   - 如果不是，则可以提交工单获取阿里云技术支持。
2. 把Demo中的混淆配置拷贝到开发者工程的混淆配置中。

如何添加硬编黑名单，硬解白名单？

- 添加硬编码黑名单

```
/**
* 添加硬编码黑名单，model和versions的顺序必须对应起来
* 黑名单内的机型将使用软编，黑名单外的机型都使用硬编
* @param models 机型Model信息列表{@link Build#MODEL}
* @param versions 系统版本号列表{@link Build.VERSION#SDK_INT}，如果不需要适配版本号，填写0即可
*/NativeAdaptiveUtil.encoderAdaptiveList(String[] models,int[] versions);
```

- 添加硬解码白名单

```
/**
* 添加硬解码器白名单，model和versions的顺序必须对应起来
* 如果开启硬解码了，则白名单中的机型将使用硬解码，白名单外的机型将使用软解码
* @param models 机型Model信息列表{@link Build#MODEL}
* @param versions 系统版本号列表{@link Build.VERSION#SDK_INT}，如果不需要适配版本号，填写0即可
* @see #setHWDecoderEnable(boolean)
*/
NativeAdaptiveUtil.decoderAdaptiveList(String[] models, int[] versions);
```

Android基础版本提示java.lang.NoSuchFieldError: No field height of type I in class Lcom/aliyun/snap/snap_core/R$id; or its superclasses (declaration of 'com.aliyun.snap.snap_core.R$id' appears in /data/app/com.rablive.jwrablive-2/base.apk:classes2.dex)，该如何解决？

问题现象：Android基础版本提示java.lang.NoSuchFieldError: No field height of type I in class Lcom/aliyun/snap/snap_core/R$id; or its superclasses (declaration of 'com.aliyun.snap.snap_core.R$id' appears in /data/app/com.rablive.jwrablive-2/base.apk:classes2.dex)。

可能原因：出现这个错误是因为在开发者的工程中存在和AAR（SDK）一样的xml，于是导致冲突。

解决方案：找到出现冲突的xml，开发者自行加前缀。目前发现容易冲突的xml包括：`activity_setting.xml` 和 `activity_video_play.xml` 。

Android基础版提示java.lang.NoSuchFieldError: No static field notification_template_lines of type I in class Lcom/aliyun/snap/snap_core/R$layout; or its superclasses (declaration of 'com.aliyun.snap.snap_core.R$layout' appears/data/app/com.Aliyun.AliyunVideoSDK.VodSaaSDemo_android-1/base.apk)，该如何解决？

目前主要原因是基础版SDK的UI不开源，所以内部是引用了support包的，但是打成AAR时是没有将support包打入的，这就导致ID不对应的情况。目前需要您将support版本包对应，如下：

```
//重要：如果工程中引入第三方库也引入了support包，也必须要保证第三方的包版本对应，建议以源码引入第三方库
compile 'com.android.support:appcompat-v7:24.2.1'
compile 'com.android.support:design:24.2.1'
```

但是有时第三方库带support包修改起来比较麻烦，且可能有些第三方包并不是源码引入的，此时建议在Application里面的gradle文件中配置。

```
configurations.all {
    resolutionStrategy {
        force 'com.android.support:appcompat-v7:24.2.1'
        force 'com.android.support:design:24.2.1'
    }
}
```

Demo使用时提示Please invoke the FileDownloader#init in Application#onCreate first，该怎么解决？

需要在Application OnCreate中调用DownloaderManager.getInstance().init(Context context);。

SDK内部是否有获取视频封面的接口？

Android短视频SDK目前专业版提供了AliyunIThumbnailFetcher接口，可以获取非关键帧的图片，其他版本建议使用系统函数取帧。

视频录制

录制如何添加普通动图？

添加普通动图需要使用AliyunIRecorder#addPaster(EffectPaster effectPaster,float sx,float sy,float sw,float sh,float rotation,boolean flip)接口，并且在EffectPaster对象中填入信息，EffectPaster的isTrack一定要设置为false，否则贴图将作为人脸贴图处理，跟随人脸变化，如果没有人脸，贴图会不显示。另外该接口必须在RecordCallback#OnInitReady()回调之后调用，否则将不会显示贴图。

如何设置录制角度？

目前设置录制角度有setRotation和setRecordRotation两个接口，setRotation接口是一个自适应接口，只要nin把手机角度传感器返回的角度值传给这个接口，就可以得到正确的录制角度和人脸角度。setRecordRotation接口是自定义的视频角度接口，可以根据您的需求定制任意角度值。

如何实现横屏录制？

- 您如果需要默认横屏只需要将界面UI元素旋转引导横屏拍摄视频，不需要设置界面横屏，让界面固定竖屏即可。

```
android:screenOrientation="portrait"
```

- 横屏拍摄的视频录制完成合成的视频是会带旋转角度的，旋转角度以录制的第一段为准。
- 如果是专业版，在编辑之后调用合成接口合成的视频将会输出一个不带角度的视频。比如原始视频为360/640，角度：270变为640 /*360，由于基础版和标准版只有录制功能，如果横屏拍摄会得到一个带旋转角度的视频，这个视频是以拍摄时第一段的角度为准的。专业版录制时的行为同基础版和标准版，合成完成后视频不带角度，转换为一个角度为0，宽高变换的视频。

关键接口函数：

```
/**
* 设置视频旋转角度值
* @param rotation
*/
void setRotation(int rotation);
```

接口调用条件：设置旋转角度需要在初始化完成之后设置，且需要保证录制第一段之前调用。

调用操作步骤：让界面固定竖屏，然后设置旋转角度即可。

1. 设置界面竖屏，让界面的UI元素旋转，引导您拍摄横屏的视频。
2. 同普通录制的初始化。
3. 在调用开始录制前调用，注意旋转角度需要您自己获取，可以参考Demo使用OrientationDetector来获取方向。

```
mRecorder.setRotation(int rotation);
```

4. 继续录制步骤，注意每次调用startRecording前都需要设置旋转角度以此来确定每段视频的旋转角度。

拍摄添加背景音乐，完成后调用finishRecordForEditor，音乐没有合成进去，是什么原因？

添加背景音乐后，必须调用finishRecording接口，才会把音乐合成进去，否则不会合成进去，造成进到编辑界面无音乐的结果。finishRecording和finishRecordForEditor之间的区别如下：

- finishRecording有两个作用，一是录制多段的时候，调用该接口可以将多段拼接成一个mp4，也就是录制指定的输出文件，二是添加了背景音乐后，调用该接口会把背景音乐合成进这个输出mp4中，无论是多段还是单段都可以。
- finishRecordForEditor不会拼接多段视频，也不会将背景音乐合成到输出mp4文件中，而是仅仅将录制的片段（startRecording > stopRecording之后就会生成一个片段），按照指定的格式配置到project.json文件中（创建Editor时传入的Uri就是该文件的Uri）。

如果添加了背景音乐的录制多段，该如何进到编辑界面呢？

先调用finishRecording将视频拼接成输出地址的mp4，然后将该mp4用AliyunIImport接口导入到编辑界面。

录制界面添加普通动图显示效果不完整，该怎么解决？

需要获取普通动图素材的宽和高。

```
AliyunIRecorder#addPaster(EffectPaster effectPaster,float sx,float sy,float sw,float sh,flo
at rotation,boolean flip)
```

其中sw和sh参数需要符合素材宽高和屏幕宽高的比列。

例如：素材的宽高分别为200、300，屏幕的宽高分别为540、720。则 `sw = (float)200/540, sh = (float)300/720` 。sx、sy是归一化的屏幕比例参数，坐标是以资源的中心点作为锚点的。

视频编辑

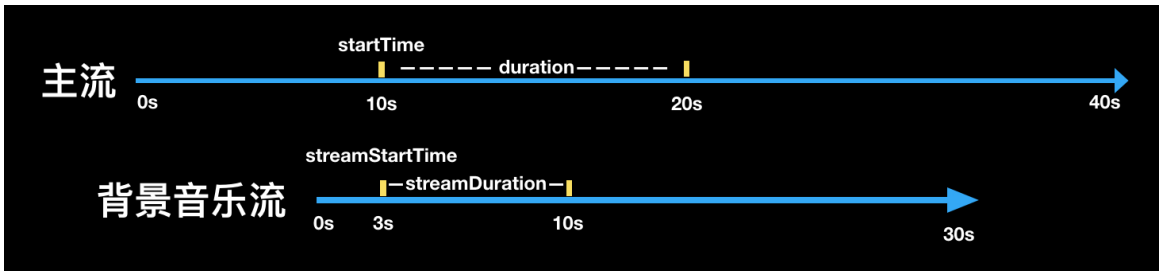
编辑时添加特效，使用AliyunICompose.compose合成出来的视频不带特效，是什么原因？

V3.5.0及以前的版本，要把编辑预览的特效持久化到本地的配置文件中，需要调用AliyunIEditor.onPause接口，如果没有调用，则特效配置不会持久化到本地文件中，那么通过AliyunICompose接口反序列化生成的Project就不带特效，也就导致合成出来的视频不带特效。

编辑时，调用applyMusic。添加音乐后，设置了startTime，为什么每次音乐流都从0开始播，而不是从startTime开始？

查看接口文档EffectBean类里对startTime的解释，startTime指的是特效在主流上的作用时间，并不是指素材流的起始时间，在V3.6.0版本之后增加了一个streamStartTime参数，这个参数是表示素材流的起始时间，即有这种需求的开发者，在V3.6.0及以后的版本中可以通过接口参数配置实现，但是V3.6.0之前的版本，只能由开发者先对素材流做裁剪，然后用裁剪的音乐去做背景音乐。

如下图所示：在播放到10s的时候，背景音乐开始播放，从音乐流的3s处播放，播放到10s处，重新从3s处开始播放，当第二遍播放到音乐流的6s时，刚好对应主流的20s位置，停止播放背景音乐。



如何添加gif作为主体流？

V3.7.0以下版本，gif格式的文件是作为图片类型导入；V3.7.0及以上版本，gif格式的文件如果作为视频类型导入，则当做视频播放gif所有帧，如果作为图片类型导入，则作为图片播放第一帧。

为什么设置完转场或者applySourceChange之后，视频卡住不动了？

需要您在进行这些操作之后调用mAliyunIEditor.play()接口。

8.3. iOS端短视频SDK常见问题

本文主要介绍短视频SDK在iOS端的使用问题。

SDK集成

导入短视频SDK时，控制台提示category方法未找到，该如何解决？

在工程target中选择Build Setting > other linker flags，添加-ObjC。

Debug包和Release包有什么区别？

Debug包包含模拟器和真机版本，可以保证模拟器编译通过，本地调试时推荐使用Debug包；Release包只包含真机版本，在提交App Store时必须使用Release包，因为Apple要求动态库提交不能包含模拟器版本。

导入短视频SDK后运行crash，提示image not found，该如何解决？

短视频SDK使用了动态库，导入动态库需要在Embedded Binaries中添加对应的framework，详细操作请参考集成SDK。

短视频SDK支持bitcode吗？

短视频SDK不支持bitcode，需要在设置中把**Enable Bitcode**设为**NO**。打包如果出现failed to verify bitcode错误，需要取消勾选**rebuild for bitcode**选项。

提示[NSDictionary oss_dictionaryWithXMLData:]: unrecognized selector sent to class，是什么原因？

未导入上传SDK，短视频SDK需要依赖上传SDK的 `AliyunOSSiOS.framework` ，详细操作请参见集成SDK。

视频录制

普通录制完成后，获取不到视频，该如何解决？

需要在录制完成回调函数 `- (void)recorderDidFinishRecording;` 里才能获取到视频。

如何实现横屏录制？

录制时候设置 `cameraRotate` 角度值，录制的视频方向会以第一段视频的角度值为准。

录制过程中更换音乐，没有生效，是什么原因？

录制过程中不支持更换音乐。

如何实现全屏录制方案？

录制分辨率9：16显示有以下两种方案：

- 方案一：和短视频SDK提供的demo中一致，iphone X上下留黑边。
- 方案二：可以调整view布局上下撑满，左右一部分内容不显示。

视频裁剪

裁剪提示1008错误，该如何解决？

关闭 `shouldOptimize` 选项。

裁剪提示700004错误，该如何解决？

输出路径未设置，请设置输出路径，可通过AliyunCrop的 `outputPath` 参数设置。

如何实现没有黑边的裁剪？

根据原始分辨率，做一个缩放，缩放后的分辨率保证是偶数。

如何裁剪一段音乐？

裁剪参数 `videoSize` 和 `ouputSize` 都无需设置，其他操作和裁剪视频时的参数保持一致。

视频编辑

编辑完成后，合成crash，出现报错提示[null length]，该如何解决？

检查水印路径（AEPGlobalWatermarkTrack的source参数）是否设置正确。

调用音量接口出现破音，是什么原因？

音量默认值100代表原声，大于100可能会破音，建议保证音量值设置在0~100。

滤镜、MV等资源找不到，该如何解决？

资源拷贝到项目中需要用folder方式导入才能保证层级关系，注意在xcode中显示的文件夹是蓝色的。

导入视频，提示operation not permit，该如何解决？

从系统相册导入的视频，需要调用系统接口获取相册访问权限，同时保证对应的AVAseset没有被销毁。

加入音频后，无法调节音量和添加、删除音效，该如何解决？

注意加入的音频需要为pcm、mp3等格式的音频文件，不能为视频文件。

视频上传

上传报错，提示没有授权，该如何解决？

问题现象：上传报错，提示没有授权。

可能原因：通过STS方式上传时所使用的STS授权已到期。

解决方案：重新获取STS授权后，再次上传。获取STS授权的流程简单介绍如下：

1. 使用阿里云主账号创建用户，给用户授予AliyunSTSAssumeRoleAccess权限。
2. 使用用户创建角色，给角色授予VODFULL权限。
3. 通过调用STS的SDK，获取STS，参考：创建角色并进行STS临时授权。

> **说明** 调用STS的SDK中的AK必须是RAM用户的AK。更多关于账号、RAM用户及授权的信息请参见账号和授权。

4. 修改policy为点播的VODFULL权限。

```
String policy = "{\n" +
          "    \"Version\": \"1\", \n" +
          "    \"Statement\": [\n" +
          "        {\n" +
          "            \"Action\": [\n" +
          "                \"vod:*\"\n" +
          "            ], \n" +
          "            \"Resource\": [\n" +
          "                \"*\" \n" +
          "            ], \n" +
          "            \"Effect\": \"Allow\"\n" +
          "        }\n" +
          "    ]\n" +
          "}"
```

上传过程中断网，为什么没有失败回调？

上传过程中断网，会自动重试，如果不想走重试接口，可以手动调用取消上传的接口。

上传后的视频，通过服务端SDK下载的视频格式为什么是m3u8？

转码配置里面，如果勾选了hls选项，则会生成m3u8格式的视频。

上传App Store

提交App Store审核时报错，该如何解决？

问题现象：提示Invalid CFBundleSupportedPlatforms value错误。

问题原因：4.3.1和3.15.0版本的QuCore-ThirdParty会存在上述问题。QuCore-ThirdParty中info.plist的CFBundleSupportedPlatforms支持的平台为iPhoneSimulator。

解决方案：

- 方法一：将info.plist中CFBundleSupportedPlatforms下支持的平台值改为iPhoneOS。
- 方法二：使用其他QuCore-ThirdParty的版本，推荐版本如下表所示。

QuCore-ThirdParty版本	短视频SDK版本
≥ 4.3.2（推荐）	≥ v3.26.0
4.3.1（不推荐）	v3.26.0 ~ v3.29.0
4.3.0	v3.24.0 ~ v3.25.1
3.15.0	< v3.24.0

8.4. AlivcFFmpeg版本依赖

本文为您介绍短视频SDK的AlivcFFmpeg版本依赖。

背景信息

如果App同时引入短视频SDK和播放器SDK（不包含AlivcFFmpeg版本），请确保引入正确的AlivcFFmpeg版本，否则会导致短视频或播放器的功能使用失败。

Android端App

AlivcFFmpeg版本	短视频SDK版本	播放器SDK版本	备注
com.aliyun.video.android:AlivcFFmpeg:4.3.2.1	≥ v3.27.0	≥ v5.4.5.0	共用
com.aliyun.video.android:AlivcFFmpeg:4.3.2.0-part	≥ v3.27.0	不适用	短视频SDK专用（较小）
com.aliyun.video.android:AlivcFFmpeg:4.3.2	≥ v3.26.0	≥ v5.4.4.0	共用
com.aliyun.video.android:AlivcFFmpeg:4.3.1.1-part	≥ v3.27.0	不适用	短视频SDK专用（较小）

AlivcFFmpeg版本	短视频SDK版本	播放器SDK版本	备注
com.aliyun.video.android:AlivcFFmpeg:4.3.1-part	≥ v3.26.0	不适用	短视频SDK专用（较小）
com.aliyun.video.android:AlivcFFmpeg:4.3.1	≥ v3.26.0	≥ v5.4.2.0	共用
com.aliyun.video.android:AlivcFFmpeg:4.3.0-part	≥ v3.25.0	不适用	短视频SDK专用（较小）
com.aliyun.video.android:AlivcFFmpeg:4.3.0	≥ v3.24.0	≥ v5.4.2.0	共用
com.aliyun.video.android:AlivcFFmpeg:2.0.1	v3.22.0、v3.23.0	[v5.1.2, v5.4.2.0)	共用
com.aliyun.video.android:AlivcFFmpeg:2.0.0	≤ v3.21.0	≤ v4.7.4	共用

iOS端App

AlivcFFmpeg版本	短视频SDK版本	播放器SDK版本	备注
pod 'QuCore-ThirdParty', '4.3.2	≥ v3.27.0	≥ v5.4.4.0	共用
pod 'QuCore-ThirdParty', '4.3.1	≥ v3.26.0	≥ v5.4.2.0	共用
pod 'QuCore-ThirdParty', '4.3.0	≥ v3.24.0	≥ v5.4.2.0	共用
pod 'QuCore-ThirdParty', '3.15.0	< v3.24.0	< v5.4.2.0	共用