短视频

视频编辑

产品文档

腾讯云

文档目录

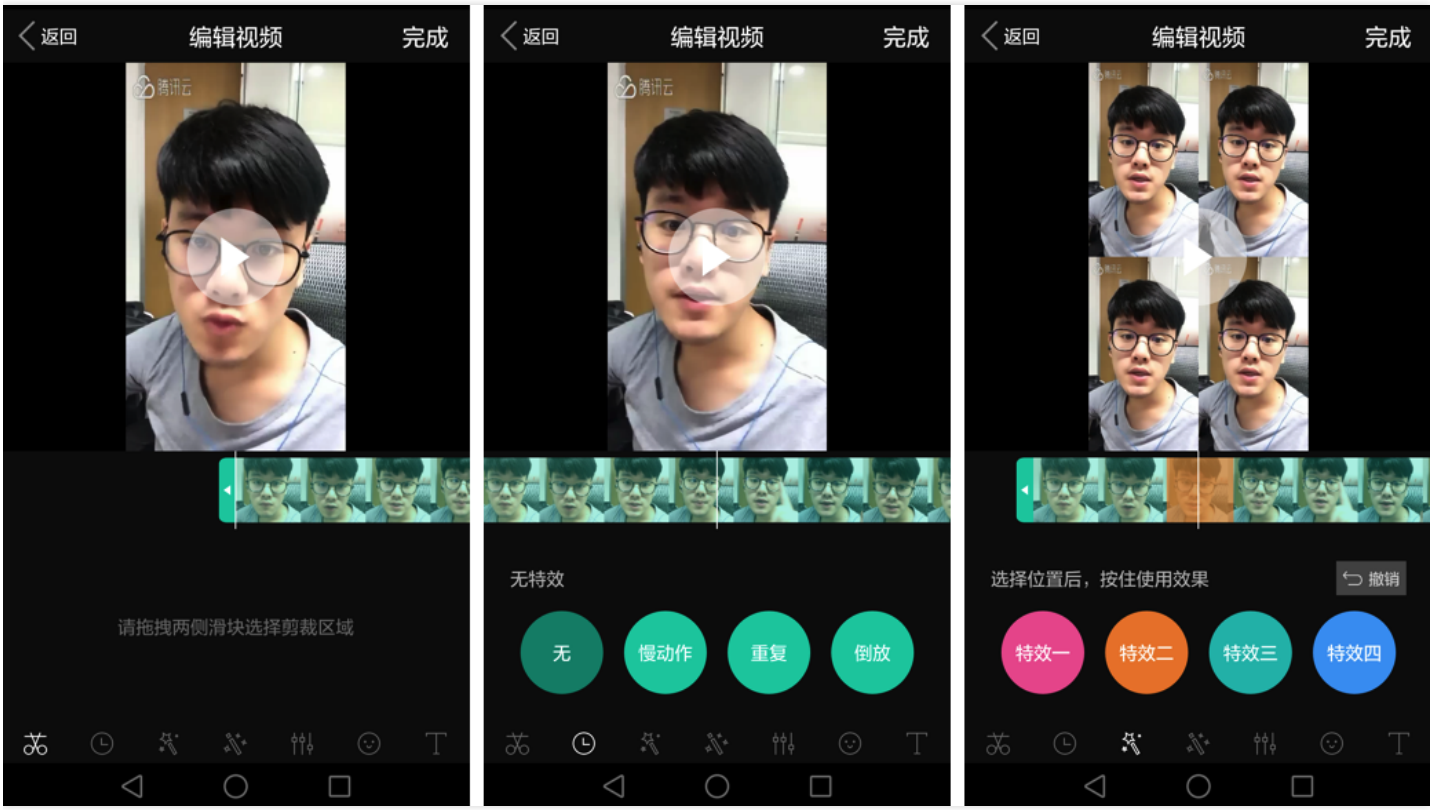
视频编辑
视频编辑（iOS）
视频编辑（Android）

视频编辑
视频编辑（iOS）

最近更新时间：2018-06-15 20:27:59

功能概览

视频编辑包括视频裁剪、时间特效（慢动作、倒放、重复）、滤镜特效（动感光波，暗黑幻影，灵魂分出窍，画面分裂）、滤镜风格（唯美，粉嫩，蓝调等）、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能，我们在 SDK 开发包的 Demo 中实现了一套 UI 源码供使用参考及体验，各功能的界面如下:



- 图 1 是视频裁剪操作界面
- 图 2 是时间特效操作界面
- 图 3 是动态滤镜操作界面
- 图 4 是美颜滤镜操作界面
- 图 5 是背景音乐操作界面
- 图 6 是静态与动态贴纸操作界面

- 图 7 是气泡字幕操作界面

编译运行 Demo 体验，从资源下载处下载SDK开发包，解压出来运行 Demo 工程RTMPiOSDemo.xcodeproj，在运行起来后的主界面中点选短视频特效即可选择视频进入进行编辑功能体验。

复用现有UI

视频编辑具有比较复杂的交互逻辑，这也决定了其 UI 复杂度很高，所以我们比较推荐复用 SDK 开发包中的 UI 源码，使用时从 Demo 中拷贝以下文件夹到自己的工程:

1. VideoEditor(代码)
2. Resource下的VideoEditor与FilterResource.bundle(资源)
3. Common 与 Third(整个 Demo 依赖的公共代码，根据自己的需要再进行删改)

UI 源码说明
源码中各个界面组件比较独立，您可根据自己产品需求再对界面进行修改加工。
主要界面组件说明：

1. VideoCutView: 视频裁剪界面，包含了视频裁剪、时间特效，滤镜特效添加等功能，如需定制裁剪界面可对此类进行修改或替换。
2. FilterSettingView: 视频添加滤镜界面，目前包含 9 种滤镜，如果修改滤镜操作的界面可对此类进行修改或替换。
3. MusicMixView: 视频添加混音界面，包含音乐文件选择，音乐信息展示，音乐长度裁剪，原音与伴音音量调整等功能，如需修改混音界面可对此类进行修改或替换。
4. TextAddView: 视频添加字幕界面，此界面只包含跳转到 VideoTextViewController 的按钮，实际添加字幕的界面及操作在 VideoTextViewController 中进行，可根据需要进行修改或去除此界面。
5. VideoTextField: 字幕输入组件，包含字幕文字输入、字幕拖动、放大、旋转、删除、添加字幕背景样式等功能，VideoTextViewController 中主要利用此组件完成字幕的添加删除。
6. VideoPasterView: 贴纸输入组件，包含动态/静态贴纸输入、贴纸拖动、放大、旋转、删除等功能，VideoPasterViewController 中主要利用此组件完成贴纸的添加删除。

VideoEditViewController 只负责将这几个界面与预览界面 VideoPreview 组合起来，并结合 SDK 去处理界面间的交互及响应，如需对界面进行整体结构的调整可在此类中进行。

自己实现 UI

如果您不考虑复用我们开发包中的 UI 代码，决心自己实现 UI 部分，则可以参考如下的攻略进行对接：

1. 预览图片组

TXVideoInfoReader 的 getVideoInfo 方法可以获取指定视频文件的一些基本信息，getSampleImages 则可以获取指定数量的预览图：

```
// 获取视频文件的信息
+ (TXVideoInfo *)getVideoInfo:(NSString *)videoPath;

// 对视频文件进行预读，均匀得生成 count 张预览图片组
+ (void)getSampleImages:(int)count
videoPath:(NSString *)videoPath
progress:(sampleProcess)sampleProcess;
```

开发包中的 VideoRangeSlider 即使用了 getSampleImages 获取了 10 张缩略图来构建一个由视频预览图组成的进度条。

2. 效果预览

视频编辑提供了 **定点预览**（将视频画面定格在某一时间点）与**区间预览**（循环播放某一时间段 A<=>B 内的视频片段）两种效果预览方式，使用时需要给 SDK 绑定一个 UIView 用于显示视频画面。

- **绑定 UIView**
  TXVideoEditer 的 initWithPreview 函数用于绑定一个 UIView 给 SDK 来渲染视频画面，绑定时需要制定**自适应**与**填充**两种模式。

```
PREVIEW_RENDER_MODE_FILL_SCREEN - 填充模式，尽可能充满屏幕不留黑边，所以可能会裁剪掉一部分画面。
PREVIEW_RENDER_MODE_FILL_EDGE - 适应模式，尽可能保持画面完整，但当宽高比不合适时会有黑边出现。
```

- **定点预览**
  TXVideoEditer 的 previewAtTime 函数用于定格显示某一个时间点的视频画面。

- **区间预览**
  TXVideoEditer 的 startPlayFromTime 函数用于循环播放某一时间段A<=>B内的视频片段。

3. 视频裁剪

视频编辑类操作都符合同一个操作原则：即先设定操作指定，最后用 generateVideo 将所有指令顺序执行，这种方式可以避免多次重复压缩视频引入的不必要的质量损失。

```
TXVideoEditer* _ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
// 设置裁剪的 起始时间 和 结束时间
[_ugcEdit setCutFromTime:_videoRangeSlider.leftPos toTime:_videoRangeSlider.rightPos];
```

```
// ...
// 生成最终的视频文件
_ugcEdit.generateDelegate = self;
[_ugcEdit generateVideo:VIDEO_COMPRESSED_540P videoOutputPath:_videoOutputPath];
```

输出时指定文件压缩质量和输出路径，输出的进度和结果会通过 generateDelegate 以回调的形式通知用户。

4. 滤镜特效

您可以给视频添加滤镜效果，例如美白、浪漫、清新等滤镜，demo提供了9种滤镜选择，同时也可以设置自定义的滤镜。
设置滤镜的方法为：

```
- (void) setFilter:(UIImage *)image;
```

其中 image 为滤镜映射图，image 设置为 nil，会清除滤镜效果。

Demo示例：

```
TXVideoEditer *_ugcEdit;
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

5. 音轨处理

您可以为视频添加自己喜欢的背景音乐，并且可以选择音乐播放的起始时间和结束时间，如果音乐的播放时间段小于视频的时间段，音乐会循环播放至视频结束。除此之外，您也可以设置视频声音和背景声音的大小，来达到自己想要声音合成效果。

设置背景音乐的方法为：

```
- (void) setBGM:(NSString *)path result:(void(^)(int))result;
- (void) setBGMAsset:(AVAsset *)asset result:(void(^)(int))result;
```

其中 path 为音乐文件路径, asset为音乐属性,从系统媒体库loading出来的音乐，可以直接传入对应的音乐属性，会极大的降低音乐从系统媒体库loading的时间，具体请参考demo用法。

设置背景音乐的开始和结束方法为：

```
- (void) setBGMStartTime:(float)startTime endTime:(float)endTime;
```

其中 startTime 为音乐起始时间，endTime 为音乐结束时间。

设置背景音乐是否循环播放方法为：

```
- (void) setBGMLoop:(BOOL)isLoop;
```

其中 isLoop 为音乐是否循环播放。

设置背景音乐在视频的添加的起始位置方法为：

```
- (void) setBGMAtVideoTime:(float)time;
```

其中 time 为音乐在视频添加的起始位置。

设置视频和背景声音大小的方法为：

```
- (void) setVideoVolume:(float)volume;
- (void) setBGMVolume:(float)volume;
```

其中 volume 表示声音的大小，取值范围 0 ~ 1 ，0 表示静音，1 表示原声大小。

Demo 示例：

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:@"defalut.mp3"];
[_ugcEdit setBGM:_BGMPath result:^(int result) {
if (result == 0) {
[_ugcEdit setBGMStartTime:0 endTime:10];
[_ugcEdit setBGMVolume:1];
[_ugcEdit setVideoVolume:1];
}
}];
```

6. 设置全局水印

您可以为视频设置水印图片，并且可以指定图片的位置

设置水印的方法为：

```
- (void) setWaterMark:(UIImage *)waterMark normalizationFrame:(CGRect)normalizationFrame;
```

其中 waterMark 表示水印图片，normalizationFrame 是相对于视频图像的归一化frame，frame 的 x，y，width，height 的取值范围都为 0~1。

Demo 示例：

```
UIImage *image = [UIImage imageNamed:@"watermark"];
[_ugcEdit setWaterMark:image normalizationFrame:CGRectMake(0, 0, 0.3 , 0.3 * image.size.height / image.size.width)];//水印大小占视频宽度的30%，高度根据宽度自适应
```

7. 设置片尾水印

您可以为视频设置片尾水印，并且可以指定片尾水印的位置
设置片尾水印的方法为：

```
- (void) setTailWaterMark:(UIImage *)tailWaterMark normalizationFrame:(CGRect)normalizationFrame
duration:(CGFloat)duration;
```

其中 tailWaterMark 表示片尾水印图片，normalizationFrame 是相对于视频图像的归一化frame，frame 的 x，y，width，height 的取值范围都为 0~1，
duration 水印的持续时长
Demo 示例：设置水印在片尾中间，持续时间 1s。

```
UIImage *tailWaterimage = [UIImage imageNamed:@"tcloud_logo"];
float w = 0.15;
float x = (1.0 - w) / 2.0;
float width = w * videoMsg.width;
float height = width * tailWaterimage.size.height / tailWaterimage.size.width;
float y = (videoMsg.height - height) / 2 / videoMsg.height;
[_ugcEdit setTailWaterMark:tailWaterimage normalizationFrame:CGRectMake(x,y,w,0) duration:1];
```

8. 滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持四种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置滤镜特效的方法为：

```
- (void) startEffect:(TXEffectType)type startTime:(float)startTime;
- (void) stopEffect:(TXEffectType)type endTime:(float)endTime;

//滤镜特效的类型（type参数），在常量 TXEffectType 中有定义：
typedef NS_ENUM(NSInteger,TXEffectType)
{
TXEffectType_ROCK_LIGHT, //动感光波
TXEffectType_DARK_DRAEM, //暗黑幻境
TXEffectType_SOUL_OUT, //灵魂出窍
TXEffectType_SCREEN_SPLIT,//视频分裂
```

```
TXEffectType_WIN_SHADOW, //百叶窗
TXEffectType_GHOST_SHADOW,//鬼影
TXEffectType_PHANTOM, //幻影
TXEffectType_GHOST, //幽灵
TXEffectType_LIGHTNING, //闪电
TXEffectType_MIRROR, //镜像
TXEffectType_ILLUSION, //幻觉
};

- (void) deleteLastEffect;
- (void) deleteAllEffect;
```

调用 deleteLastEffect() 删除最后一次设置的滤镜特效。

调用 deleteAllEffect() 删除所有设置的滤镜特效。

Demo 示例：

在 1-2s 之间应用第一种滤镜特效；在 3-4s 之间应用第2种滤镜特效；删除 3-4s 设置的滤镜特效

```
//在1-2s之间应用第一种滤镜特效
[_ugcEdit startEffect(TXEffectType_SOUL_OUT, 1.0);
[_ugcEdit stopEffect(TXEffectType_SOUL_OUT, 2.0);
//在3-4s之间应用第2种滤镜特效
[_ugcEdit startEffect(TXEffectType_SPLIT_SCREEN, 3.0);
[_ugcEdit stopEffect(TXEffectType_SPLIT_SCREEN, 4.0);
//删除3-4s设置的滤镜特效
[_ugcEdit deleteLastEffect];
```

9. 慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
- (void) setSpeedList:(NSArray *)speedList;

//TXSpeed 的参数如下：
@interface TXSpeed: NSObject
@property (nonatomic, assign) CGFloat startTime; //加速播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //加速播放结束时间(s)
@property (nonatomic, assign) TXSpeedLevel speedLevel; //加速级别
@end

// 目前支持变速速度的几种级别，在常量 TXSpeedLevel 中有定义：
typedef NS_ENUM(NSInteger, TXSpeedLevel) {
SPEED_LEVEL_SLOWEST, //极慢速
SPEED_LEVEL_SLOW, //慢速
SPEED_LEVEL_NOMAL, //正常速
```

```
SPEED_LEVEL_FAST, //快速
SPEED_LEVEL_FASTEST, //极快速
};
```

Demo 示例：

```
// SDK拥有支持多段变速的功能。 此DEMO仅展示一段慢速播放
TXSpeed *speed =[[TXSpeed alloc] init];
speed.startTime = 1.0;
speed.endTime = 3.0;
speed.speedLevel = SPEED_LEVEL_SLOW;
[_ugcEdit setSpeedList:@[speed]];
```

10. 倒放

您可以将视频画面倒序播放，设置倒放的方法：

```
- (void) setReverse:(BOOL)isReverse;
```

Demo 示例：

```
[_ugcEdit setReverse:YES];
```

11. 重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。
设置重复片段方法：

```
- (void) setRepeatPlay:(NSArray *)repeatList;

//TXRepeat 的参数如下：
@interface TXRepeat: NSObject
@property (nonatomic, assign) CGFloat startTime; //重复播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //重复播放结束时间(s)
@property (nonatomic, assign) int repeatTimes; //重复播放次数
@end
```

Demo 示例：

```
TXRepeat *repeat = [[TXRepeat alloc] init];
repeat.startTime = 1.0;
repeat.endTime = 3.0;
repeat.repeatTimes = 3; //重复次数
[_ugcEdit setRepeatPlay:@[repeat]];
```

12. 静/动态贴纸

您可以为视频设置静态贴纸或者动态贴纸。

设置静态贴纸的方法：

```
- (void) setPasterList:(NSArray *)pasterList;

// TXPaster 的参数如下：
@interface TXPaster: NSObject
@property (nonatomic, strong) UIImage* pasterImage; //贴纸图片
@property (nonatomic, assign) CGRect frame; //贴纸frame（注意这里的frame坐标是相对于渲染view的坐标）
@property (nonatomic, assign) CGFloat startTime; //贴纸起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //贴纸结束时间(s)
@end
```

设置动态贴纸的方法：

```
- (void) setAnimatedPasterList:(NSArray *)animatedPasterList;

// TXAnimatedPaster 的参数如下：
@interface TXAnimatedPaster: NSObject
@property (nonatomic, strong) NSString* animatedPasterpath; //动图文件路径
@property (nonatomic, assign) CGRect frame; //动图的frame（注意这里的frame坐标是相对于渲染view的坐标）
@property (nonatomic, assign) CGFloat rotateAngle; //动图旋转角度 (0 ~ 360)
@property (nonatomic, assign) CGFloat startTime; //动图起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //动图结束时间(s)
@end
```

Demo示例：

```
- (void)setVideoPasters:(NSArray*)videoPasterInfos
{
NSMutableArray* animatePasters = [NSMutableArray new];
NSMutableArray* staticPasters = [NSMutableArray new];
for (VideoPasterInfo* pasterInfo in videoPasterInfos) {
if (pasterInfo.pasterInfoType == PasterInfoType_Animate) {
TXAnimatedPaster* paster = [TXAnimatedPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.rotateAngle = pasterInfo.pasterView.rotateAngle * 180 / M_PI;
```

```
paster.animatedPasterpath = pasterInfo.path;
[animatePasters addObject:paster];
}
else if (pasterInfo.pasterInfoType == PasterInfoType_static){
TXPaster *paster = [TXPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.pasterImage = pasterInfo.pasterView.staticImage;
[staticPasters addObject:paster];
}
}
[_ugcEditer setAnimatedPasterList:animatePasters];
[_ugcEditer setPasterList:staticPasters];
}
```

如何自定义动态贴纸？

动态贴纸的本质是：将 **一串图片**，按照 **一定的顺序** 以及 **时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

封装格式

以 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK内部将获取到该动态贴纸对应的 config.json，并且按照 json 中定义的格式进行动态贴纸的展示。

注：该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸

13. 气泡字幕

您可以为视频添加字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置字幕的方法为：

```
- (void) setSubtitleList:(NSArray *)subtitleList;

TXSubtitle 的参数如下：
@interface TXSubtitle: NSObject
@property (nonatomic, strong) UIImage* titleImage; //字幕图片（这里需要客户把承载文字的控件转成image图片）
@property (nonatomic, assign) CGRect frame; //字幕的frame（注意这里的frame坐标是相对于渲染view的坐标）
@property (nonatomic, assign) CGFloat startTime; //字幕起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //字幕结束时间(s)
@end
```

其中：

titleImage ： 表示字幕图片，如果上层使用的是 UILabel 之类的控件，请先把控件转成 UIImage ，具体方法可以参照 demo 的示例代码。

frame ： 表示字幕的 frame ，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的frame ，具体可以参照 demo 的示例代码。

startTime ： 字幕作用的起始时间。

endTime ： 字幕作用的结束时间。

因为字幕这一块的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现，可以大大降低您的接入成本。

Demo示例：

```
@interface VideoTextInfo : NSObject
@property (nonatomic, strong) VideoTextFiled* textField;
@property (nonatomic, assign) CGFloat startTime; //in seconds
@property (nonatomic, assign) CGFloat endTime;
@end

videoTextInfos = @[VideoTextInfo1, VideoTextInfo2 ...];

for (VideoTextInfo* textInfo in videoTextInfos) {
TXSubtitle* subtitle = [TXSubtitle new];
subtitle.titleImage = textInfo.textField.textImage; //UILabel（UIView） -> UIImage
subtitle.frame = [textInfo.textField textFrameOnView:_videoPreview]; //计算相对于渲染view的坐标
```

```
subtitle.startTime = textInfo.startTime; //字幕起始时间
subtitle.endTime = textInfo.endTime; //字幕结束时间
[subtitles addObject:subtitle]; //添加字幕列表
}

[_ugcEditer setSubtitleList:subtitles]; //设置字幕列表
```

如何自定义气泡字幕？

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

注：以上单位均为px

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 .json 格式封装：

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

注：该封装格式用户可以自行决定，非SDK强制性要求

字幕过长？

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并？
我们在Demo中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。
您也可以修改相关控件源代码，来满足自身的业务要求。

14. 图片编辑

SDK在4.7版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画，BGM，贴纸等效果。
接口函数如下：

```
/*
*pitureList:转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考demo用法），否则内存占用可能过大，导致编辑过程异常）
*fps: 转场图片生成视频后的fps（15 ~ 30）
* 返回值：
* 0 设置成功；
* -1 设置失败，请检查图片列表是否存在，图片数量是否大于等于3张，fps是否正常；
*/
- (int)setPictureList:(NSArray<UIImage *> *)pitureList fps:(int)fps;

/*
*transitionType:转场类型，详情见 TXTransitionType
* 返回值：
* duration 转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
- (void)setPictureTransition:(TXTransitionType)transitionType duration:(void(^)(CGFloat))duration;
```

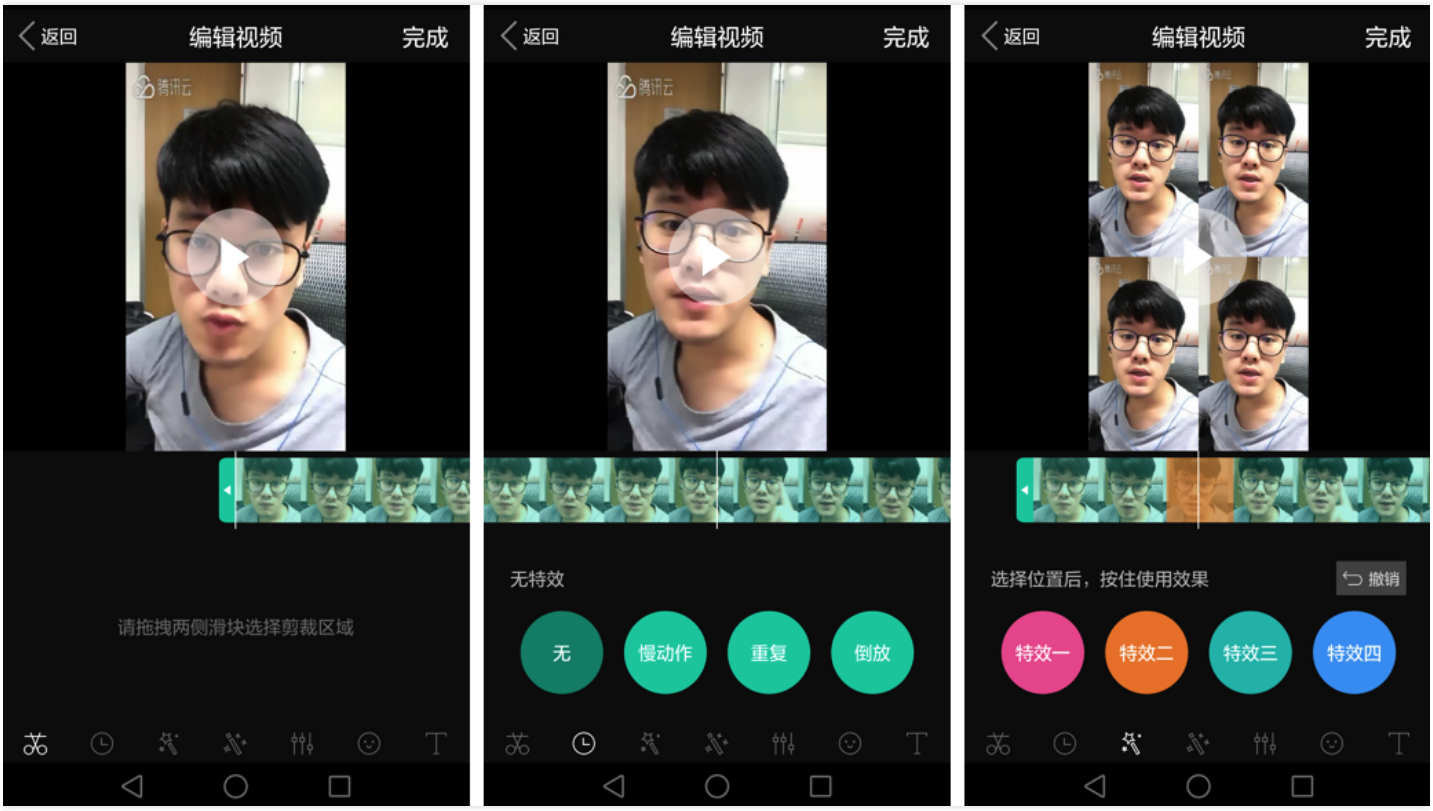
其中，setPictureList接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。setPictureTransition接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过duration获取转场的时长。

图片编辑暂不支持的功能：重复，倒放，快速/慢速，片尾水印，其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

视频编辑（Android）

最近更新时间：2018-08-17 16:49:52

视频编辑包括视频裁剪、时间特效（慢动作、倒放、重复）、滤镜特效（动感光波，暗黑幻影，灵魂出窍，画面分裂）、滤镜风格（唯美，粉嫩，蓝调等）、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能，我们在 SDK 开发包的 Demo 中实现了一套 UI 源码供使用参考及体验，各功能的界面如下:



- 图 1 是视频裁剪操作界面
- 图 2 是时间特效操作界面
- 图 3 是动态滤镜操作界面
- 图 4 是美颜滤镜操作界面
- 图 5 是背景音乐操作界面
- 图 6 是静态与动态贴纸操作界面

- 图 7 是气泡字幕操作界面

编译运行 Demo 体验，从资源下载处下载 Android完整版开发包，解压出来运行 Demo 工程，在运行起来后的主界面中点选视频编辑即可选择视频进入进行编辑功能体验。
Android 编辑的代码位置：com.tencent.liteav.demo.shortvideo包名下面，其中 choose.TCVideoChooseActivity 为本地视频列表界面，editor.TCVideoPreprocessActivity 为视频预处理界面，TCVideoEditerActivity 为视频编辑界面，editor.bgm 包下为背景音的实现，editor.bubble 包下为气泡字幕的实现，editor.cutter 包下为裁剪实现，editor.filter 包下为静态滤镜实现，editor.motion 包下为滤镜动效实现，editor.paster 包下为动态/静态贴纸实现，editor.time 包下为时间特效的实现。

复用现有 UI

视频编辑具有比较复杂的交互逻辑，这也决定了其 UI 复杂度很高，所以我们比较推荐复用 SDK 开发包中的 UI 源码，使用时从Demo中拷贝以下文件夹到自己的工程:

1. shortvideo/editor 下的代码
2. res/下的VideoEditor 资源
3. jniLibs/ 下的 jar 和 so
4. 在 AndroidManifest.xml 中声明这几个 Activity，并声明权限

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

<activity
android:name=".shortvideo.choose.TCVideoChooseActivity"
android:screenOrientation="portrait"
/>
<activity
android:name=".shortvideo.editor.TCVideoPreprocessActivity"
android:launchMode="singleTop"
android:screenOrientation="portrait" />
<activity
android:name=".shortvideo.editor.TCVideoEditerActivity"
android:screenOrientation="portrait">
<activity
android:name=".shortvideo.editor.paster.TCPasterActivity"
android:screenOrientation="portrait" />
<activity
android:name=".shortvideo.editor.bubble.TCWordEditActivity"
```

```
android:windowSoftInputMode="adjustResize|adjustPan"
android:screenOrientation="portrait" />
```

自己实现 UI

如果您不考虑复用我们开发包中的 UI 代码，决心自己实现 UI 部分，则可以参考如下的攻略进行对接：

1. 视频导入

快速导入

快速导入视频，可以直接观看到视频编辑的预览效果，支持视频裁剪、时间特效（慢动作）、滤镜特效、滤镜风格、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能，不支持的功能有时间特效（重复、倒放）。

快速导入生成精准缩略图，开发包中的 TCVideoEditerActivity 即使用了此方法获取了 10 张缩略图来构建一个由视频预览图组成的进度条。

```
TXVideoEditConstants.TXThumbnail thumbnail = new TXVideoEditConstants.TXThumbnail();
thumbnail.count = TCVideoEditerWrapper.mThumbnailCount; //设置缩略图张数
thumbnail.width = 100; //缩略图宽
thumbnail.height = 100;//缩略图高

mTXVideoEditer.setThumbnail(thumbnail);
mTXVideoEditer.setThumbnailListener(mThumbnailListener); //设置缩略图回调

mTXVideoEditer.processVideo(false);

private TXVideoEditer.TXThumbnailListener mThumbnailListener = new TXVideoEditer.TXThumbnailL
istener() {
@Override
public void onThumbnail(int index, long timeMs, final Bitmap bitmap) {
// 动态生成缩略图的回调，每生成一张，返回一张Bitmap
}
};
```

全功能导入

全功能导入，支持所有的功能，包括时间特效（重复、倒放）。需要为视频先预处理操作。

经过全功能导入后的视频可以精确的 seek 到每个时间点，看到对应的画面，预处理操作同时还可以精确的生成当前时间点视频缩略图。

1、生成精确缩略图的方法：

```
public void setThumbnail(TXVideoEditConstants.TXThumbnail thumbnail);

//TXThumbnail参数如下：
public final static class TXThumbnail{
public int count; // 缩略图个数
public int width; // 缩略图宽
public int height; // 缩略图高
}
```

注意：

- 生成精确缩略图**setTumbnail**方法必须在**processVideo**方法调用之前
- 缩略图的宽高最好不要设置视频宽高，SDK内部缩放效率更高

2、进行预处理的方法：

```
//预处理方法
public void processVideo();

//设置视频预处理回调
public void setVideoProcessListener(TXVideoProcessListener listener);
```

Demo示例：

```
int thumbnailCount = (int) mTXVideoEditer.getTXVideoInfo().duration / 1000; //根据视频时长生成缩略图个数

TXVideoEditConstants.TXThumbnail thumbnail = new TXVideoEditConstants.TXThumbnail();
thumbnail.count = thumbnailCount;
thumbnail.width = 100;
thumbnail.height = 100;

mTXVideoEditer.setThumbnail(thumbnail); //设置预处理生成的缩略图
mTXVideoEditer.setThumbnailListener(mThumbnailListener); // 缩略图回调

mTXVideoEditer.setVideoProcessListener(this); //视频预处理进度回调
mTXVideoEditer.processVideo(); //进行预处理
```

2. 视频基本信息

TXVideoInfoReader 的 **getVideoFileInfo** 方法可以获取指定视频文件的一些基本信息

```
// 获取视频文件的信息
getVideoFileInfo(String videoPath){...}
```

3. 效果预览

视频编辑提供了**区间预览**（播放某一时间段A<=>B内的视频片段）预览方式，使用时需要给 SDK 绑定一个 FrameLayout 用于显示视频画面。

- **绑定 FrameLayout**
  TXVideoEditer 的 initWithPreview 函数用于绑定一个 FrameLayout 给 SDK 来渲染视频画面，绑定时需要制定**自适应**与**填充**两种模式。

  PREVIEW_RENDER_MODE_FILL_SCREEN - 填充模式，尽可能充满屏幕不留黑边，所以可能会裁剪掉一部分画面。
  PREVIEW_RENDER_MODE_FILL_EDGE - 适应模式，尽可能保持画面完整，但当宽高比不合适时会有黑边出现。

- **区间预览**
  TXVideoEditer 的 startPlayFromTime 函数用于播放某一时间段 A<=>B 内的视频片段。

4. 视频裁剪

视频编辑类操作都符合同一个操作原则：即先设定操作指定，最后用 generateVideo 将所有指令顺序执行，这种方式可以避免多次重复压缩视频引入的不必要的质量损失。

```
mTXVideoEditer.initWithPreview(param);
// 设置裁剪的 起始时间 和 结束时间
mTXVideoEditer.setCutFromTime(mEditPannel.getSegmentFrom(), mEditPannel.getSegmentTo());
// ...
// 生成最终的视频文件
mTXVideoEditer.setVideoGenerateListener(this);
mTXVideoEditer.generateVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mVideoOutputPath);
```

输出时指定文件压缩质量和输出路径，输出的进度和结果会通过 `TXVideoEditer.TXVideoGenerateListener` 以回调的形式通知用户。

> 注意：输出文件路径请在外部新建一个空文件，传入绝对路径，不要和原视频的路径相同。

5. 美颜滤镜

您可以给视频添加滤镜效果，例如美白、浪漫、清新等滤镜，demo 提供了 9 种滤镜选择，同时也可以设置自定义的滤镜。
设置滤镜调用 **TXVideoEditer** 的 **setFilter** 方法：

```
void setFilter(Bitmap bmp)
```

其中 Bitmap 为滤镜映射图，bmp 设置为 null，会清除滤镜效果。

```
void setSpecialRatio(float specialRatio)
```

该接口可以调整滤镜程度值，一般为0.0 ~ 1.0。

```
void setFilter(Bitmap leftBitmap, float leftIntensity, Bitmap rightBitmap, float rightIntensity, float leftRatio)
```

该接口能够实现组合滤镜，即左右可以添加不同的滤镜。leftBitmap为左侧滤镜、leftIntensity为左侧滤镜程度值；rightBitmap为右侧滤镜、rightIntensity为右侧滤镜程度值；leftRatio为左侧滤镜所占的比例,一般为0.0 ~ 1.0。当leftBitmap或rightBitmap为null，则该侧清除滤镜效果。

6. 音轨处理

您可以为视频添加自己喜欢的背景音乐，并且可以选择音乐播放的起始时间和结束时间，如果音乐的播放时间段小于视频的时间段，音乐会循环播放至视频结束。除此之外，您也可以设置视频声音和背景声音的大小，来达到自己想要声音合成效果。

设置背景音乐的方法为：

```
public int setBGM(String path);
```

其中 mBGMPath 为音乐文件路径，返回值为 0 表示设置成功； 其他表示失败，如：不支持的音频格式。

设置 BGM 起止时间的方法为：

```
public void setBGMStartTime(long startTime, long endTime);
```

其中起止时间的单位均为毫秒。

设置视频和背景声音大小的方法为：

```
public void setVideoVolume(float volume);
public void setBGMVolume(float volume);
```

其中 volume 表示声音的大小， 取值范围 0 ~ 1 ， 0 表示静音， 1 表示原声大小。

设置从视频的某一个位置起开始添加背景音乐

```
public void setBGMAtVideoTime(long videoStartTime);
```

设置背景音乐是否循环播放：true-循环播放，false-不循环播放

```
public void setBGMLoop(boolean looping);
```

Demo示例：

```
mTXVideoEditer.setBGM(mBGMPath);
mTXVideoEditer.setBGMStartTime(startTime, endTime);
mTXVideoEditer.setBGMVolume(0.5f);
mTXVideoEditer.setVideoVolume(0.5f);
```

7. 设置水印

您可以为视频设置水印图片，并且可以指定图片的位置

设置水印的方法为：

```
public void setWaterMark(Bitmap waterMark, TXVideoEditConstants.TXRect rect);
```

其中 waterMark 表示水印图片，rect 是相对于视频图像的归一化 frame，frame 的 x，y，width，height 的取值范围都为 0 ~ 1。

Demo示例：

```
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXVideoEditer.setWaterMark(mWaterMarkLogo, rect);
```

8. 设置片尾水印

您可以为视频设置片尾水印，并且可以指定片尾水印的位置。
设置片尾水印的方法为：

```
setTailWaterMark(Bitmap tailWaterMark, TXVideoEditConstants.TXRect txRect, int duration);
```

其中 tailWaterMark 表示片尾水印图片，txRect 是相对于视频图像的归一化 txRect，txRect 的 x, y, width 取值范围都为0~1，duration 水印的持续时长，单位秒。

Demo 实例：设置水印在片尾中间，持续 3 秒

```
Bitmap tailWaterMarkBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.tcloud_logo);
TXVideoEditConstants.TXRect txRect = new TXVideoEditConstants.TXRect();
txRect.x = (mTXVideoInfo.width - tailWaterMarkBitmap.getWidth()) / (2f * mTXVideoInfo.width);
txRect.y = (mTXVideoInfo.height - tailWaterMarkBitmap.getHeight()) / (2f * mTXVideoInfo.height);
txRect.width = tailWaterMarkBitmap.getWidth() / (float) mTXVideoInfo.width;
mTXVideoEditer.setTailWaterMark(tailWaterMarkBitmap, txRect, 3);
```

9. 单帧预览

经过预处理后的视频可以精确的seek到每个时间点的方法：

```
public void previewAtTime(long timeMs);
```

10. 滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持四种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置滤镜特效的方法为：

```
public void startEffect(int type, long startTime);
public void stopEffect(int type, long endTime)

//滤镜特效的类型（type参数），在常量 TXVideoEditConstants 中有定义：
TXEffectType_SOUL_OUT - 滤镜特效1
TXEffectType_SPLIT_SCREEN - 滤镜特效2
TXEffectType_DARK_DRAEM - 滤镜特效3
TXEffectType_ROCK_LIGHT - 滤镜特效4

public void deleteLastEffect();
public void deleteAllEffect();
```

调用 deleteLastEffect() 删除最后一次设置的滤镜特效。

调用 deleteAllEffect() 删除所有设置的滤镜特效。

Demo 示例：

在1-2s 之间应用第一种滤镜特效；在 3-4s 之间应用第 2 种滤镜特效；删除 3-4s 设置的滤镜特效

```
//在1-2s之间应用第一种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 1000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 2000);
```

```
//在3-4s之间应用第2种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 3000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 4000);
//删除3-4s设置的滤镜特效
mTXVideoEditer.deleteLastEffect();
```

11. 慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
public void setSpeedList(List speedList)；

//TXSpeed 的参数如下：
public final static class TXSpeed {
public int speedLevel; // 变速级别
public long startTime; // 开始时间
public long endTime; // 结束时间
}

// 目前支持变速速度的几种级别，在常量 TXVideoEditConstants 中有定义：
SPEED_LEVEL_SLOWEST - 极慢
SPEED_LEVEL_SLOW - 慢
SPEED_LEVEL_NORMAL - 正常
SPEED_LEVEL_FAST - 快
SPEED_LEVEL_FASTEST - 极快
```

Demo 示例：

```
// SDK拥有支持多段变速的功能。 此 DEMO 仅展示一段慢速播放
List list = new ArrayList<>(1);
TXVideoEditConstants.TXSpeed speed = new TXVideoEditConstants.TXSpeed();
speed.startTime = startTime; // 开始时间
speed.endTime = mTXVideoEditer.getTXVideoInfo().duration; // 结束时间
speed.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; // 慢速
// 添加到分段变速中
list.add(speed);

mTXVideoEditer.setSpeedList(list);
```

12. 倒放

您可以将视频画面倒序播放。通过调用 **setReverse(true)** 开启倒序播放，调用 **setReverse(false)** 停止倒序播放。

注意：**setTXVideoReverseListener()** 老版本(SDK4.5以前)首次监听是否倒放完成在新版本无需调用即可生效。

Demo示例：

```
mTXVideoEditer.setTXVideoReverseListener(mTxVideoReverseListener);
mTXVideoEditer.setReverse(true);
```

13. 重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。目前 Android 只支持设置一段画面重复，重复三次。
如需取消之前设置的重复片段，调用 **setRepeatPlay(null)** 即可。

设置重复片段方法：

```
public void setRepeatPlay(List repeatList);

//TXRepeat 的参数如下：
public final static class TXRepeat {
public long startTime; //重复播放起始时间(ms)
public long endTime; //重复播放结束时间(ms)
public int repeatTimes; //重复播放次数
}
```

Demo 示例：

```
long currentPts = mVideoProgressController.getCurrentTimeMs();

List repeatList = new ArrayList<>();
TXVideoEditConstants.TXRepeat repeat = new TXVideoEditConstants.TXRepeat();
repeat.startTime = currentPts;
repeat.endTime = currentPts + DEAULT_DURATION_MS;
repeat.repeatTimes = 3; //目前只支持重复三次
repeatList.add(repeat); //目前只支持重复一段时间
mTXVideoEditer.setRepeatPlay(repeatList);
```

14. 静/动态贴纸

您可以为视频设置静态贴纸或者动态贴纸。
设置静态贴纸的方法：

```
public void setPasterList(List pasterList);

// TXPaster 的参数如下：
public final static class TXPaster {
public Bitmap pasterImage; // 贴纸图片
public TXRect frame; // 贴纸的frame（注意这里的frame坐标是相对于渲染view的坐标）
```

```
public long startTime; // 贴纸起始时间(ms)
public long endTime; // 贴纸结束时间(ms)
}
```

设置动态贴纸的方法：

```
public void setAnimatedPasterList(List animatedPasterList);

// TXAnimatedPaster 的参数如下：
public final static class TXAnimatedPaster {
public String animatedPasterPathFolder; // 动态贴纸图片地址
public TXRect frame; // 动态贴纸frame(注意这里的frame坐标是相对于渲染view的坐标 )
public long startTime; // 动态贴纸起始时间(ms)
public long endTime; // 动态贴纸结束时间(ms)
public float rotation;
}
```

Demo示例：

```
List animatedPasterList = new ArrayList<>();
List pasterList = new ArrayList<>();
for (int i = 0; i < mTCLayerViewGroup.getChildCount(); i++) {
PasterOperationView view = (PasterOperationView) mTCLayerViewGroup.getOperationView(i);
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX();
rect.y = view.getImageY();
rect.width = view.getImageWidth();
TXCLog.i(TAG, "addPasterListVideo, adjustPasterRect, paster x y = " + rect.x + "," + rect.y);

int childType = view.getChildType();
if (childType == PasterOperationView.TYPE_CHILD_VIEW_ANIMATED_PASTER) {
TXVideoEditConstants.TXAnimatedPaster txAnimatedPaster = new TXVideoEditConstants.TXAnimate
dPaster();

txAnimatedPaster.animatedPasterPathFolder = mAnimatedPasterSDcardFolder + view.getPasterNam
e() + File.separator;
txAnimatedPaster.startTime = view.getStartTime();
txAnimatedPaster.endTime = view.getEndTime();
txAnimatedPaster.frame = rect;
txAnimatedPaster.rotation = view.getImageRotate();

animatedPasterList.add(txAnimatedPaster);
TXCLog.i(TAG, "addPasterListVideo, txAnimatedPaster startTimeMs, endTime is : " + txAnimatedPaster.
startTime + ", " + txAnimatedPaster.endTime);
} else if (childType == PasterOperationView.TYPE_CHILD_VIEW_PASTER) {
```

```
TXVideoEditConstants.TXPaster txPaster = new TXVideoEditConstants.TXPaster();

txPaster.pasterImage = view.getRotateBitmap();
txPaster.startTime = view.getStartTime();
txPaster.endTime = view.getEndTime();
txPaster.frame = rect;

pasterList.add(txPaster);
TXCLog.i(TAG, "addPasterListVideo, txPaster startTimeMs, endTime is : " + txPaster.startTime + ", " + tx
Paster.endTime);
}
}

mTXVideoEditer.setAnimatedPasterList(animatedPasterList); //设置动态贴纸
mTXVideoEditer.setPasterList(pasterList); //设置静态贴纸
```

如何自定义动态贴纸？

动态贴纸的本质是：将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

封装格式

以 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK内部将获取到该动态贴纸对应的config.json，并且按照json中定义的格式进行动态贴纸的展示。

注：该封装格式为SDK内部强制性要求，请严格按照该格式描述动态贴纸

15. 气泡字幕

您可以为视频设置气泡字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置气泡字幕的方法为：

```
public void setSubtitleList(List subtitleList);

//TXSubtitle 的参数如下：
public final static class TXSubtitle {
public Bitmap titleImage; // 字幕图片
public TXRect frame; // 字幕的frame
public long startTime; // 字幕起始时间(ms)
public long endTime; // 字幕结束时间(ms)
}

public final static class TXRect {
public float x;
public float y;
public float width;
}
```

其中：
titleImage ： 表示字幕图片，如果上层使用的是 TextView 之类的控件，请先把控件转成 Bitmap ，具体方法可以参照 demo 的示例代码。
frame ： 表示字幕的 frame ，注意这个frame是相对于渲染 view（initWithPreview时候传入的view）的frame ，具体可以参照 demo 的示例代码。
startTime ： 字幕作用的起始时间。
endTime ： 字幕作用的结束时间。

因为字幕的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现， 可以大大降低您的接入成本。

Demo示例：

```
mSubtitleList.clear();
for (int i = 0; i < mWordInfoList.size(); i++) {
TCWordOperationView view = mOperationViewGroup.getOperationView(i);
```

```
TXVideoEditConstants.TXSubtitle subTitle = new TXVideoEditConstants.TXSubtitle();
subTitle.titleImage = view.getRotateBitmap(); //获取Bitmap
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX(); // 获取相对parent view的x坐标
rect.y = view.getImageY(); // 获取相对parent view的y坐标
rect.width = view.getImageWidth(); // 图片宽度
subTitle.frame = rect;
subTitle.startTime = mWordInfoList.get(i).getStartTime(); // 设置开始时间
subTitle.endTime = mWordInfoList.get(i).getEndTime(); // 设置结束时间
mSubtitleList.add(subTitle);
}
mTXVideoEditer.setSubtitleList(mSubtitleList); // 设置字幕列表
```

如何自定义气泡字幕？

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

注：以上单位均为px

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 .json 格式封装

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

注：该封装格式用户可以自行决定，非SDK强制性要求

字幕过长？

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并？

我们在 Demo 中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。

您也可以修改相关控件源代码，来满足自身的业务要求。

16. 自定义视频输出

设置最终生成视频的压缩分辨率和输出路径

```
public void generateVideo(int videoCompressed, String videoOutputPath)
```

参数videoCompressed在TXVideoEditConstants中可选常量

```
VIDEO_COMPRESSED_360P ——压缩至360P分辨率（360*640）
VIDEO_COMPRESSED_480P ——压缩至480P分辨率（640*480）
VIDEO_COMPRESSED_540P ——压缩至540P分辨率 (960*540)
VIDEO_COMPRESSED_720P ——压缩至720P分辨率 (1280*720)
```

如果源视频的分辨率小于设置的常量对应的分辨率，按照原视频的分辨率；
如果源视频的分辨率大于设置的常量对象的分辨率，进行视频压缩至相应分辨率

目前支持自定义视频的码率，这里建议设置的范围 600-12000kbps，如果设置了这个码率，SDK最终压缩视频时会优先选取这个码率，注意码率不要太大或太小，码率太大，视频的体积会很大，码率太小，视频会模糊不清。

```
public void setVideoBitrate(int videoBitrate);
```

17. 图片编辑

SDK在4.9版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画，BGM，贴纸等效果。
接口函数如下：

```
/*
* bitmapList:转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考demo用法），否则内存占用可能过大，导致编辑过程异常）
* fps: 转场图片生成视频后的fps（15 ~ 30）
* 返回值：
* 0 设置成功；
* -1 设置失败，请检查图片列表是否存在
*/
public int setPictureList(List<Bitmap> bitmapList, int fps);

/*
* type:转场类型，详情见 TXVideoEditConstants
* 返回值：
```

```
* duration 转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
public long setPictureTransition(int type)
```

- 其中，setPictureList接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过返回值获取转场的时长。
- 需要注意接口调用顺序，先调用setPictureList，再调用setPictureTransition。
- 图片编辑暂不支持的功能：重复，倒放，快速/慢速，其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

18. 释放

当您不再使用mTXVideoEditer对象时，一定要记得调用 **releasee()** 释放它。

19、根据时间点获取缩略图

输出指定时间点列表的缩略图，可以根据返回的多张缩略图生成一个GIF封面

```
List<Long> list = new ArrayList<>();
list.add(10000L);
list.add(12000L);
list.add(13000L);
list.add(14000L);
list.add(15000L);

TXVideoEditer txVideoEditer = new TXVideoEditer(TCVideoPreviewActivity.this);
txVideoEditer.setVideoPath(mVideoPath);
txVideoEditer.setThumbnailListener(new TXVideoEditer.TXThumbnailListener() {
@Override
public void onThumbnail(int index, long timeMs, Bitmap bitmap) {
Log.i(TAG, "bitmap:" + bitmap + ",timeMs:" + timeMs);
saveBitmap(bitmap, timeMs);
}
});
txVideoEditer.getThumbnailList(list, 200, 200);
```

注意

- List中时间点不能超出视频总时长，对于超出总时长的返回最后一张图片
- 设置的时间点单位是毫秒(ms)