播放器 SDK

高级功能

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

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

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

高级功能
- Web 高级功能
  - VR 播放插件（TCPlayerVRPlugin）
  - 安全检查插件（TCPlayerSafeCheckPlugin）
- 移动端高级功能
  - 发布日志
  - 短视频组件（TUIPlayerShortVideo）
    - Android
    - iOS
  - 画中画组件（TUIPIP）
    - iOS
  - VR 播放插件
  - 终端极速高清
- 播放质量监控
  - 点播播放数据
  - SDK 质量监控

高级功能
Web 高级功能
VR 播放插件（TCPlayerVRPlugin）

最近更新时间：2024-04-01 20:55:21

TCPlayerVRPlugin 插件可用于 VR 全景视频播放，播放中可以通过陀螺仪转动或手势操作来改变视角，提供多种属性和方法来控制播放表现，支持 PC 端和移动端。

使用条件

- 目前 Web 播放器 SDK 5.0.0 以上版本支持使用 VR 播放插件。
- VR 播放需获取 Web 端播放器高级版 License 方可使用。

接入方式

播放器初始化过程参见 TCPlayer 集成指引 和 API 文档。
初始化播放器实例时，可通过声明 plugins 插件的方法开启 VR 播放能力，开启后播放器会自动加载并使用本插件：

```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器ID，必须与html中一致
  plugins: {
    VR: {
      isEnableController: true,
      ...
    },
  }
});
```

VR 插件属性说明

名称	说明	默认值
isEnableController	是否打开 VR 控制器	true
isEnableZoom	是否允许画面缩放	true
yaw	初始化左右视角角度，单位为度	0
pitch	初始化上下视角角度，单位为度	0
fov	初始化可视视角，单位为度	65
yawRange	限制视角活动的范围	[-180, 180]
pitchRange	限制视角活动的范围	[-90, 90]
fovRange	限制视角活动的范围	[30, 110]

版权所有：腾讯云计算（北京）有限责任公司

VR 插件方法说明

VR 插件初始化后，会生成实例，实例化之后会进入 VR 模式播放视频，可以在播放器实例上找到 VR 实例，通过 VR 实例可以调用相关方法：

– lookAt

在一段时间通过动画形式移动到特定角度的视角。

```
player.plugins['VR'].lookAt({ yaw: 30 }, 1000);
```

– setGyroMode

如果您的设备拥有运动传感器（陀螺仪、加速传感器），您可以通过设备的运动来改变视角，这个方法可取值为 `'VR' | 'none' | 'yawPitch'` 。

```
player.plugins['VR'].setGyroMode('none');
```

– enableSensor

获取使用运动传感器的权限。一般在 Android 设备，默认会开启运动传感器，在 iOS 13+，则需要触发人机交互手动获取权限。

```
player.plugins['VR'].enableSensor();
```

说明：

1. 在浏览器劫持播放的环境，无法支持 VR 视频的播放。
2. Android 端播放器初始化后会默认进入 VR 模式，并开启陀螺仪。
3. iOS 端根据系统版本不同，表现会有差异：
   3.1 系统版本13+，需要手动点击页面，触发人机交互并获取权限后，才能打开陀螺仪。
   3.2 系统版本12.2 – 13，需进入系统设置手动开启运动传感器。通常来说开启路径一般为设置 > Safari > 动作与方向访问，开启传感器后刷新页面，即可打开。

示例

单击 此处 可参考示例。

安全检查插件（TCPlayerSafeCheckPlugin）

最近更新时间：2024-09-04 09:35:31

TCPlayerSafeCheckPlugin 插件用于检测播放环境和播放状态是否正常，保护播放安全。需结合播放器 TCPlayer 使用。

使用条件

- 目前 Web 播放器 SDK 5.0.0 以上版本支持使用 TCPlayerSafeCheckPlugin 播放插件。
- TCPlayerSafeCheckPlugin 需获取 Web 端播放器高级版 License 方可使用。

在长期维护播放器的过程中，遭遇过多种攻击方式，对以下可通过第三方工具盗取视频资源的行为，本插件从以下三个方面进行了针对性防范：

1. MSE 环境检测

部分浏览器插件或脚本可以劫持当前播放环境，通过修改 Media Source Extensions API (MSE) 来截获播放数据，并最终实现下载视频，本插件可以检测并防范此类攻击手段。

2. 安全结构检查

通过第三方工具或脚本可以修改播放器结构，达到去除播放标识、去除水印等目的，并实现录屏，本插件会检测播放器结构是否被侵入篡改，一旦发现此类行为，会及时中止播放。

3. 接口响应完整性校验

播放器使用过程中，需要与云点播服务端进行数据交互，一旦接口数据被修改，会对正常的播放行为产生影响。本插件可以检测并防范此类攻击手段。

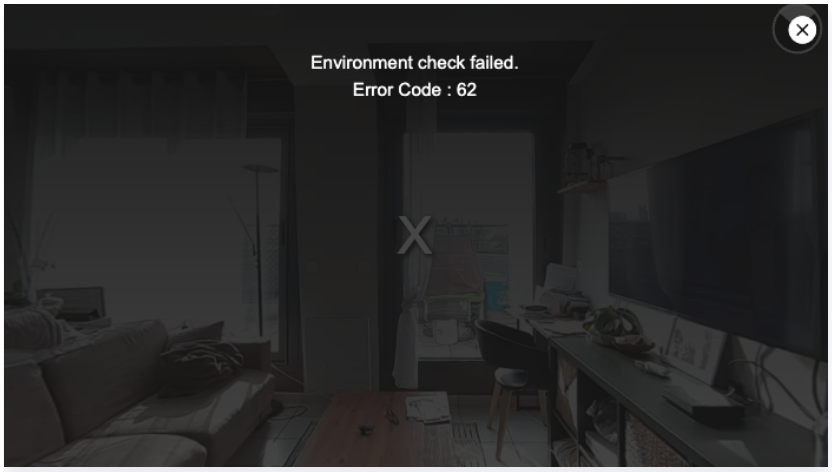
使用方式

集成 TCPlayer 可以参见 TCPlayer 集成指引 和 API 文档 。
创建播放器实例时，可通过声明 plugins 插件的方法开启安全检查能力，开启后播放器会自动加载并使用本插件：

```
const player = TCPlayer('player-container-id', {
  plugins: {
    SafeCheck: true,
  }
});
```

效果

开启插件后，播放器会自动检测环境是否安全，如果遭遇上述攻击手段会自动终止播放，并进行相应提示，如下图：



插件会提示的错误码如下：

错误码	说明
60	安全结构检查异常
61	接口响应完整性异常
62	MSE 环境检测异常

移动端高级功能
发布日志

最近更新时间：2025-05-14 15:43:02

TUI 组件

Android 端

TUI 短视频 2.1.6 @ 2025.05.14

修复部分直播声音异常问题。

TUI 短视频 2.1.5 @ 2025.03.04

修复数据列表操作时候的已知问题。

TUI 短视频 2.1.4 @ 2025.01.22

- 优化起播流程，进一步减少起播时间，优化列表流畅度，减少卡顿。
- 修复部分页面回收后不显示首帧画面的问题。
- 设置数据前，pause 也会生效，直到视频起播或者调用了 resume。
- 预下载并发数量默认修改为1个，来减少弱网情况下的网络负载压力。
- TUIShortVideoView 新增 pause(retainPaused) 方法，retainPaused 传 true 之后，不管出现任何状况都会一直维持当前的 pause 状态，直到调用了 resume。
- 修复其余已知问题。

TUI 短视频 2.1.0 @ 2024.12.30

- 图层中播放器新增 setVodSubtitleDataListener 字幕文本数据回调方法。
- 日志默认为打开状态。
- 设置数据前，pause 也会生效，直到视频起播或者调用了 resume。
- TUIShortVideoView 的 setListener 新增可选实现方法 onVodPlayerReady。
- 调用 resume 后不再立刻回调 onPlayBegin，而是以播放器实际状态为准。
- 新增拉伸平铺模式。
- 增加 license 容错机制。
- 修复其余已知问题。

TUI 短视频 2.0.9 @ 2024.11.26

- Demo 中 TUIVideoInfoLayer 划走之后不再默认隐藏。
- 修复点播暂停下进行 seek，后续无法续播的问题。
- 修复其余已知问题。

TUI 短视频 2.0.8 @ 2024.11.21

- 修复 TUIVideoSource 无法序列化的问题。

版权所有：腾讯云计算（北京）有限责任公司

- UI 组件中 TUIShortVideoListener 新增可选方法滑动回调监听 onPageScrolled。
- 修复其余已知问题。

TUI 短视频 2.0.7 @ 2024.11.06

- 点播新增弱网重试能力。
- 新增全局视频预下载状态监听能力。
- TUIShortVideoView 支持下拉刷新和上拉加载的辅助计算方法。
- TUIShortVideoView 支持自定义动画效果。
- 修复其余已知问题。

TUI 短视频 2.0.3 @ 2024.07.12

- 新增 TUIShortVideoView#setModels 第二个参数支持传入初始位置下标。
- 修复滑动过程中，偶然出现的画面显示问题，首帧显示异常问题，Layer 组件焦点丢失问题。
- 修复填充完数据后，调用 TUIShortVideoView#startPlayForIndex 回调两次 TUIShortVideoListener#onPageChanged 的问题。
- 修复填充完跳跃到自定义页面，上一个播放器无法停止的问题。
- 修复调整平铺模式、镜像、渲染角度不生效问题。
- 修复 ITUIVodPlayer#setStartTime 不生效和进度异常问题。
- 修复已知的内存泄漏问题。
- 优化列表滑动流畅度。

TUI 短视频 2.0.1 @ 2024.05.08

- 新增直播、自定义页面能力。
- 新增支持视频跨页面无缝切换。
- 新增支持超分、音量均衡。
- 优化对外 API。
- 修复问题、优化性能。

TUI 短视频 1.3.0 @ 2024.02.27

- 新增列表数据控制、滑动控制和播放循环模式配置功能。
- 补齐 Player 接口和播放事件，协议支持获取视频渲染对象属性。
- 新增通过 extViewType 自定义页面类型，支持数据变更回调。
- 新增自定义视频图层 TXCloudVideoView 功能。
- TUIBaseLayer 新增 View 释放回调。
- 优化播放缓存相关配置功能。

TUI 短视频 1.2.0 @ 2023.11.23

- 新增独立分辨率配置以及相关 demo。
- layer 中可获取当前视频分辨率相关配置。

TUI 短视频 1.1.0 @ 2023.10.27

- TUIVideoStrategy 支持 mediaType、自适应码率配置，预加载大小支持 float。
- TUIShortVideoListener 移除 onReachLast 回调，新增 onPageChanged、onNetStatus 回调。
- layer 中可获得 renderMode，TUIVideoSource 新增 extInfo 成员变量用于业务扩展，新增 isAutoPlay 控制是否自动播放。

TUI 短视频 1.0.0 @ 2023.08.17

发布1.0.0 版本短视频组件

iOS 端

TUI 短视频 2.1.0.68 @ 2025.03.11

- 增加外部设置播放器渲染位置的接口。
- 增加字幕数据回调接口。
- 优化播放器启播流程。
- 修复偶现播放器黑屏问题。
- 修复偶现播放器未自动启播的问题。

TUI 短视频 2.1.0.44 @ 2024.12.27

- 修复播放记录异常问题。
- 修复直播播放器没有声音的问题。
- 修复播放器自动播放失败的问题。
- 修复短视频列表滚动停止位置错误的问题。
- 修复插入视频滚动异常问题。
- 优化 License 鉴权逻辑。
- 优化短视频控制层及回调接口。

TUI 短视频 2.1.0.26 @ 2024.11.15

- 新增播放列表滑动代理回调和播放页卡反聚焦回调接口。
- 修复[TUIShortVideoView didScrollToCellWithIndex:animated:]接口导致的错位播放问题。
- 优化播放过程过程中滑动体验。
- 优化性能和内存占用过高问题。

TUI 短视频 2.0.3 @ 2024.07.17

- 内部不再接管前后台的 pause/resume 事件，统一由业务方接管。
- FiledID 视频换链存储逻辑优化，避免多次换链节省网络资源。
- 修复已知的内存泄漏和线程泄漏问题。
- 优化播放策略，提升性能。
- 修复其他的已知 BUG。

TUI 短视频 2.0.1 @ 2024.05.08

- 新增直播、自定义页面能力。
- 新增支持视频跨页面无缝切换。

- 新增支持超分、音量均衡。
- 优化对外 API。
- 修复问题、优化性能。

TUI 短视频 1.3.0 @ 2024.02.27

- 新增列表数据控制，滑动控制和播放循环模式配置功能。
- 补齐 Player 接口和播放事件，协议支持获取视频渲染对象属性。
- 新增通过 extViewType 自定义页面类型，支持数据变更回调。
- 优化播放缓存相关配置功能。

TUI 短视频 1.2.0 @ 2023.11.23

- 新增切换分辨率，读取分辨率相关接口。
- 预下载策略优化。

TUI 短视频 1.1.0 @ 2023.10.27

- 播放策略支持 mediaType、自适应码率配置，预加载大小支持 float。
- 播放回调支持 播放器状态，播放器进度，播放器网络状态回调，当前播放模型数据回调。
- UI 自定义能力优化。新增下拉刷新 UI 控件的用户自定义。支持独立的短视频播放策略。
- 新增 extInfo 成员变量用于业务扩展，新增 isAutoPlay 控制是否自动播放。

TUI 短视频 1.0.0 @ 2023.08.17

- 发布1.0.0 版本短视频组件

VR 播放插件

版本 1.0.0 @ 2023.08.17

新功能：
发布1.0.0 版本 VR 播放插件。

终端极速高清

版本 1.1.0 @ 2024.04.11

升级终端极速高清算法。

版本 1.0.0 @ 2023.08.02

新功能：
发布1.0.0 版本终端极速高清。

短视频组件（TUIPlayerShortVideo）Android

最近更新时间：2025-02-08 11:30:44

组件简介

TUIPlayerShortVideo 组件是腾讯云推出的一款性能优异，支持视频极速首帧和流畅滑动，提供优质播放体验的短视频组件。

- 首帧秒开：首帧时间是短视频类应用核心指标之一，直接影响用户的观看体验。短视频组件通过预播放、预下载、播放器复用和精准流量控制等技术，实现极速首帧、滑动丝滑的优质播放体验，从而提升用户播放量和停留时长。
- 优秀的性能：通过播放器复用和加载策略的优化，在保证极佳流畅度的同时，始终让内存和 CPU 消耗保持在较低的水平。
- 快速集成：组件对复杂的播放操作进行了封装，提供默认的播放 UI，同时支持 FileId 和 Url 播放，可低成本快速集成到您的项目中。

效果对比

以下视频演示了，在同等环境下，未经过优化和经过优化之后的短视频使用的对比差异。

- 优化前，可以明显感觉到视频起播的卡顿感。
- 优化后，可以达到无感起播的体验，优化后起播平均时长达到10毫秒 – 30毫秒。

未优化短视频	优化后短视频
观看视频	观看视频

TUIPlayerKit 下载

TUIPlayerKit SDK 和 Demo 可单击这里 下载。

集成 TUIPlayerShortVideo 组件

环境准备

Android 系统最低版本要求：Android SDK ⩾ 19

添加短视频需要的依赖：

```
// 如果您使用的是专业版本的SDK，则用：api
'com.tencent.liteav:LiteAVSDK_Professional:latest.release'
api 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
implementation (name:'tuiplayercore-release_x.x.x', ext:'aar')
implementation (name:'tuiplayershortvideo-release_x.x.x', ext:'aar')
implementation 'androidx.appcompat:appcompat:1.0.0'
implementation 'androidx.viewpager2:viewpager2:1.1.0-beta02'
```

> **注意：**
> 其中 `tuiplayercore-release` 和 `tuiplayershortvideo-release` 中的 `x.x.x` 为版本号，注意2个 aar 的版本号必须一致。

SDK 需要的权限：

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

设置混淆规则：
在 proguard-rules.pro 文件中，将相关类加入不混淆名单：

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

申请播放器高级版 License

使用 TUIPlayer Kit 组件需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引来获取。若您已获取对应 License，可前往 腾讯云视立方控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请移动端播放器高级版 License，将会出现视频播放失败、黑屏等现象。

设置 License

短视频组件需要设置 License 才能使用。

```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
        .licenseKey("Your license key")
        .licenseUrl("Your license url")
        .build();
TUIPlayerCore.init(context, config);
```

添加 UI 组件

在布局文件中，添加短视频 UI 组件。

```
<com.tencent.qcloud.tuiplayer.shortvideo.ui.view.TUIShortVideoView
    android:id="@+id/my_video_view"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
```

设置生命周期

设置生命周期后，组件内会根据当前 Activity 生命周期，自行对组件进行暂停、继续播放以及释放。例如在 app 退出后台的时候，短视频会自行暂停播放，当返回 app 之后，会继续播放视频。**也可不设置该周期，根据业务需要自行对短视频进行控制。**

```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

配置监听

TUIShortVideoView 的监听提供了多个回调，如下代码演示：

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
  @Override
  public void onPageChanged(int index, TUIVideoSource videoSource) {
      if (index >= mSuperShortVideoView.getCurrentDataCount() - 1) {
        // append next page data
        mSuperShortVideoView.appendModels(data);
      }
  }

  @Override
  public void onCreateVodLayer(TUIVodLayerManager layerManager, int viewType) {
      // add your vod layer to here
      layerManager.addLayer(new TUICoverLayer());
  }

  @Override
  public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
      // add your live layer to here
  }

  @Override
  public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
{
      // add your custom layer to here
  }

  // Optional
  @Override
  public void onNetStatus(TUIVideoSource model, Bundle bundle) {
  }

  // Optional
  @Override
  public void onPageScrolled(int layOutPos, float positionOffset, int
positionOffsetPixels) {
  }

  // Optional
  @Override
  public void onVodPlayerReady(ITUIVodPlayer player, TUIVideoSource source) {
  }
 });
```

当每次页面位置发生变动的时候，会回调 `onPageChanged` 方法，此处可以做类似分页加载的能力。
当列表创建布局的时候，会根据您添加数据的类型，回调 `onCreateVodLayer` 、 `onCreateLiveLayer` 或者 `onCreateCustomLayer` 方法。
其中如果setModels时，数据是 TUIVideoSource 类型的，这里将会回调 `onCreateVodLayer` ，如果是 TUILiveSource 类型的，将会回调 `onCreateLiveLayer` ，如果是继承 TUIPlaySource 实现的自定义数据类型，将会回调

`onCreateCustomLayer` 。
此处还会将 `TUIPlayerSource` 中的 `extViewType` 回调到这里，用于业务根据viewType区分不同的layer组，layer的创建会在自定义图层中讲到。
视频开始播放之后，会将当前正在播放的视频的网络状态通过 `onNetStatus` 回调出来，回调详情可查看 这里 。

填充数据

使用 `setModels` 可以设置数据，并清空掉原本的数据，视频也会从设置的数据源的第一个视频开始重新播放，使用 `appendModels` 可以追加数据，用于分页操作，**当填充完数据之后，会自动从第一个视频开始播放，如果不需要自动播放，可以将第一个视频调用 `setAutoPlay` 设置为 false。**

```
// 点播，可继承TUIVideoSource来扩展属性
TUIVideoSource videoSource = new TUIVideoSource();
videoSource.setCoverPictureUrl(model.placeholderImage);
// fileId填写，fileId和url只需要填写一个
videoSource.setAppId(model.appid);
videoSource.setPSign(model.pSign);
videoSource.setFileId(model.fileid);
// url填写
videoSource.setVideoURL(model.videoURL);
shortVideoData.add(videoSource);

// 直播，可继承TUILiveSource来扩展属性
TUILiveSource liveSource = new TUILiveSource();
// 直播资源url
liveSource.setUrl(liveUrl);
liveSource.setCoverPictureUrl(coverUrl);
shortVideoData.add(liveSource);

// 非播放器页面，例如图片、广告等，此处可根据业务需求定制不同的数据，只需要继承 TUIPlaySource
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// 设置数据
mSuperShortVideoView.setModels(shortVideoData);

// 设置初始数据的同时，可以指定初始位置， index 超过集合大小则无效
mSuperShortVideoView.setModels(shortVideoData, index);

// 如果是分页操作，可选择追加数据
mSuperShortVideoView.appendModels(shortVideoData);
```

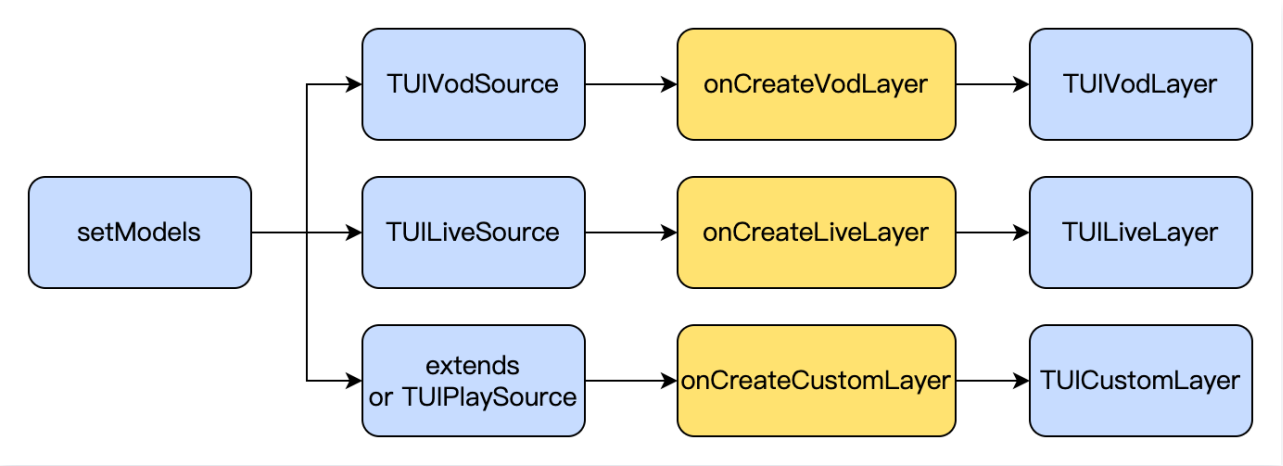
自定义图层

一、简介

1. 图层类别

如果需要对TUI短视频进行UI的定制，需要使用短视频的图层能力。

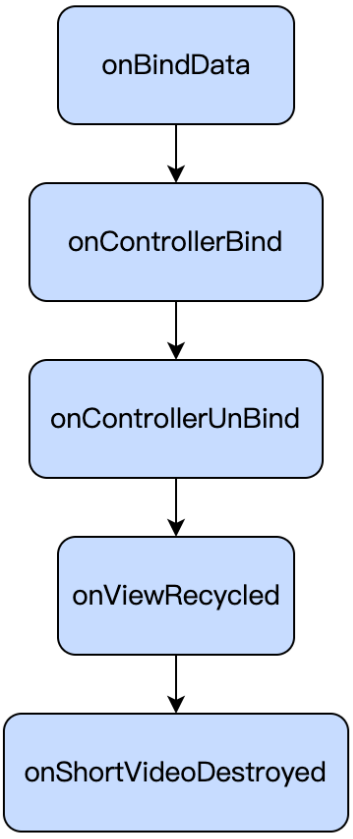
Android TUI 短视频，采用图层管理的方式，提供给每个短视频页面自定义 UI 能力。通过图层管理器，每个页面可以进行对象化管理，更好的处理视频 UI 与播放器和视频组件之间的交互。
目前有三种页面，点播、直播和自定义页面，三个页面分别对应的layer为 `TUIVodLayer` 、 `TUILiveLayer` 和 `TUICustomLayer` 。可以按照需求来继承不同的layer。不同的layer基类都针对其场景提供了符合场景的接口和回调。
对应关系如下图所示：



图层的显示和隐藏，会直接对 View 进行添加和移除，不会造成界面过度渲染。
图层会根据添加的顺序，来决定界面的展示顺序，先添加的在最上层，会优先展示，后添加的在最底层，会被之前添加的覆盖。
短视频列表由于有页面复用机制，当图层中有业务数据相关的UI展示时，需要在 `onBindData` 绑定数据的时候，对界面UI进行重置或者重新设置新值。

2. 图层生命周期

三种 Layer 的生命周期如下图所示



其生命周期含义如下：

生命周期名称	含义
onBindData	代表当前 Layer 已经绑定数据，可以在该生命周期中，做一些静态UI的数据初始化工作。
onControllerBind	该页面已经是当前短视频列表正在展示的页面，**除了自定义页面** `TUICustomLayer` **以外**，此时调用 `getPlayer()` 和getController()将不再为空，可以对播放器、页面容器进行操作。
onControllerUnBind	页面被划走，该生命周期之后将无法再获得到播放器和videoView，该生命周期可以做一些资源的回收、界面重置工作。
onViewRecycled	页面被回收，会用到其他数据和播放器上，建议在该生命周期中重置界面所有数据，回收相关资源。
onShortVideoDestroyed	TUI 组件被销毁，即 TUIShortVideoView 被调用了 release 方法。

二、创建自定义图层

1. 创建自定义图层布局

创建自定义图层，以点播layer为例，需要继承 TUIVodLayer，然后实现自己需要的图层。

以视频详情图层为例，首先需要实现 createView 和 tag 方法。createView 是图层 view 的创建方法，tag 是业务侧用来区分图层的字符串标签。
createView 会在 layer 添加到 LayerManager 的时候被调用。

```
@Override
public View createView(ViewGroup parent) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.player_video_info_layer, parent, false);
    mSeekBar = view.findViewById(R.id.vsb_tui_video_progress);
    mTvProgress = view.findViewById(R.id.tv_tui_progress_time);
    mIvPause = view.findViewById(R.id.iv_tui_pause);
    mSeekBar.setListener(this);
    return view;
}

@Override
public String tag() {
    return "TUIVideoInfoLayer";
}
```

createView 中创建了一个 View 并返回。这里可以使用 LayoutInflater 从 XML 中加载布局，也可以使用代码直接创建布局。

2. 展示布局

View 创建完成之后，需要在合适的时机去展示。TUIBaseLayer 提供了丰富的事件回调。详细信息可以参见 图层回调 。
视频详情图层在获得数据的时候就可以展示布局了。所以在 onBindData 把图层展示出来。

```
@Override
public void onBindData(TUIVideoSource videoSource) {
    show();
}
```

3. 操作自己的组件

还可以在其他事件中对自己创建的组件进行操作，这里以点播 layer 为例，需要先将自己的组件声明为成员变量，在 `onCreateView` 中赋值，随后在播放器事件，例如暂停按钮的显示和隐藏，以及播放进度的回调，参见如下代码。

```
@Override
public void onPlayBegin() {
    super.onPlayBegin();
    if (null != mIvPause) {
        mIvPause.setVisibility(View.GONE);
    }
}

@Override
public void onPlayPause() {
    super.onPlayPause();
```

```
    if (null != mIvPause) {
        mIvPause.setVisibility(View.VISIBLE);
    }
}

@Override
public void onPlayProgress(long current, long duration, long playable) {
    videoDuration = duration;
    if (null != mSeekBar) {
        // ensure a refresh at every percentage point
        int progressInt = (int) (((1.0F * current) / duration) * 100);
        if(lastProgressInt != progressInt) {
            setProgress(progressInt / 100F);
            lastProgressInt = progressInt;
        }
    }
}
```

4. 控制播放器

除了接收来自播放器的事件之外，还可以对播放器进行控制。例如调用播放器进行进度的跳转。

```
@Override
public void onDragDone(VideoSeekBar seekBar) {
    TUIPlayerController controller = getPlayerController();
    if (null != controller && videoDuration > 0) {
        controller.seekTo((int) ((videoDuration * seekBar.getBarProgress()) /
1000));
    }
    if (null != mTvProgress) {
        mTvProgress.setVisibility(View.GONE);
    }
}
```

目前在 TUIBaseLayer 中，可以通过 getVideoView 获得当前 page 的 VideoView 对象，通过 getPlayerController 获得当前视频的播放控制器（只有当前 page 是当前短视频列表正在播放的视频的时候才会有），getPlayer 获得当前播放器对象。由于播放器和视频 view 会在滑动过程中被释放或者复用，所以这三个对象在获取的时候，都可能会获得空对象，需要进行判空。

5. 图层被回收时进行释放

在图层被回收的时候，需要进行一些释放操作。防止一些外部的对象对图层造成持有，产生了内存泄漏。

```
@Override
public void onViewRecycled(TUIBaseVideoView videoView) {
    // release your resource
}
```

6. 监听是否滑动到当前 layer

如果需要监听当前 page 是否是当前播放视频，可以对 controller 进行监听，**当触发 `onControllerBind` 的时候，该layer被controller绑定，代表该layer的页面即将展示出来开始播放，当触发 `onControllerUnBind` 的时候，controller发生解绑，代表页面被划走。**

```
@Override
public void onControllerUnBind(TUIPlayerController controller) {
    super.onControllerUnBind(controller);
    show();
}
```

以上代码，是在页面划走之后， controller 解绑，为了后续再划回来不是黑屏，触发显示封面图。

7. 点播通过 onRecFileVideoInfo 回调获取视频信息

当设置的 fileId 视频源的时候，layer 会通过 `onRecFileVideoInfo` 将视频的信息回调出来，参见如下代码：

```
@Override
public void onRecFileVideoInfo(TUIFileVideoInfo params) {
    if(isShowing()) {
        TUIBaseVideoView videoView = getVideoView();
        if (null != videoView && null != params) {
            String coverUrl = params.getCoverUrl();
            if (!TextUtils.isEmpty(coverUrl)) {
                ImageView imageView = getView();
                Glide.with(videoView).load(coverUrl)
                        .centerCrop()
                        .into(imageView);
                coverUrlFromServer = coverUrl;
            }
        }
    }
}
```

该方法会在只使用 fileID 播放的时候回调。会返回视频 URL 链接、封面图、时长、雪碧图等信息。
建议尽可能的通过 URL 传入短视频组件进行播放，并提前赋值好封面图 URL ，这样能够增加短视频加载性能。

8. 通过 onRcvFirstIframe 方法判断视频首帧是否到来

使用方法参见如下代码：

```
@Override
public void onRcvFirstIframe() {
    hidden();
}
```

例如封面图等场景，需要在收到首帧事件后来隐藏封面图。

三、管理图层

当集成短视频组件 TUIShortVideoView 后，使用 TUIShortVideoView 设置监听会在合适的时机回调 item 的创建方法 onCreateItemLayer，来添加或者管理自定义图层。

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {

    // ......

 @Override
  public void onCreateVodLayer(TUIVodLayerManager layerManager, int viewType) {
      layerManager.addLayer(new TUIVideoInfoLayer(mShortVideoView,
ShortVideoFragment.this));
      layerManager.addLayer(new TUICoverLayer());
      layerManager.addLayer(new TUILoadingLayer());
      layerManager.addLayer(new TUIErrorLayer());
  }

  @Override
  public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
      layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView,
ShortVideoFragment.this));
      layerManager.addLayer(new TUILiveLoadingLayer());
      layerManager.addLayer(new TUILiveErrorLayer());
  }

  @Override
  public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
{
      if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
          layerManager.addLayer(new PicDisplayLayer());
      }
  }
});
```

onCreateItemLayer 有两个参数，layerManager 为图层管理器，可以添加、移除、查询图层。添加方式如上图所示。`viewType` 为当前 page 的视频类型，如果您在 `TUIPlayerSource` 中自定义了 `extViewType` ，这里的 `viewType` 将会是您定义的，如果没有定义，将会根据页面类型返回 `ITEM_TYPE_VOD` 、 `ITEM_TYPE_LIVE` 或者 `ITEM_TYPE_CUSTOM` 。
如果不需要图层，可以将图层移除，**移除之后，layer中会回调 `unBindLayerManager` 方法。**

```
layerManager.removeLayer(layer);
```

如果需要获得图层的层级，做图层的交互操作，可以通过以下方法获取图层的层级：

```
layerManager.indexOfLayer(layer);
```

四、使用自定义图层创建图片展示页面

1. 实现自己的自定义数据

这里以展示图片为例，创建一个带图片链接的数据。

```
public class DemoImgSource extends TUIPlaySource {

    private String mImgUrl;

    public DemoImgSource(String imgUrl) {
        mImgUrl = imgUrl;
        // 您可以指定不同的viewType来区分自定义页面类型
        setExtViewType(SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE);
    }

    public String getImgUrl() {
        return mImgUrl;
    }

    public void setImgUrl(String imgUrl) {
        this.mImgUrl = imgUrl;
    }
}
```

2. 实现自定义页面的ui

数据实现好后，需要定制自己自定义页面的UI，以展示图片为例，需要继承 `TUICustomLayer` 实现layer。

```
public class PicDisplayLayer extends TUICustomLayer {

    private ImageView mDisplayImgView;

    @Override
    public View createView(ViewGroup parent) {
        // 创建页面view
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.tuiplayer_img_display_layer, parent,
false);
        mDisplayImgView = view.findViewById(R.id.iv_img_display);
        return view;
    }

    @Override
    public void onBindData(TUIPlaySource videoSource) {
        super.onBindData(videoSource);
        // 数据与页面发生绑定，可以拿到该页面对应的数据源
        if (videoSource.getExtViewType() ==
SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
            DemoImgSource source = (DemoImgSource) videoSource;
            Glide.with(mDisplayImgView).load(source.getImgUrl())
                    .into(mDisplayImgView);
        }
```

```
    }

    @Override
    public String tag() {
        return "PicDisplayLayer";
    }
}
```

在 TUI 短视频回调中将自己的图层添加进去

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {

   // ......

  @Override
  public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
{
     // custom layer
      if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
          layerManager.addLayer(new PicDisplayLayer());
      }
  }
});
```

3. 填充数据到 TUI 短视频

```
// 自定义，DemoImgSource继承自TUIPlaySource,自定义数据，此处可根据业务需求定制不同的数据
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// 设置数据
mSuperShortVideoView.setModels(shortVideoData);
```

随后，该自定义页面，将会根据您在列表中添加的位置，展示在短视频列表对应的页面位置上。

TUI 短视频接口

1. 配置 License

使用 TUI 组件，需要配置对应 premium 的 License，示例如下：

```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
        .licenseKey(LICENCE_KEY)
        .licenseUrl(LICENCE_URL)
        .build();
TUIPlayerCore.init(getApplicationContext(), config);
```

2. 设置生命周期监听

用于 TUIShortVideoView 的生命周期控制，内部自行根据 lifeCycle 状态，进行列表视频的暂停、播放和销毁，该接口可不设置，由业务来接管调用。

```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

3. 设置短视频监听

用于监听 TUIShortVideoView 的事件，其中包括加载分页数据时机，创建 page 的时候回调，可以在该回调添加图层。

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
    @Override
    public void onCreateVodLayer(TUIVodLayerManager layerManager, int viewType) {
        layerManager.addLayer(new TUIVideoInfoLayer(mShortVideoView,
ShortVideoFragment.this));
        layerManager.addLayer(new TUICoverLayer());
        layerManager.addLayer(new TUILoadingLayer());
        layerManager.addLayer(new TUIErrorLayer());
    }

    @Override
    public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
        layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView,
ShortVideoFragment.this));
        layerManager.addLayer(new TUILiveLoadingLayer());
        layerManager.addLayer(new TUILiveErrorLayer());
    }

    @Override
    public void onCreateCustomLayer(TUICustomLayerManager layerManager, int
viewType) {
        if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
            layerManager.addLayer(new PicDisplayLayer());
        }
    }

    @Override
    public void onPageChanged(int index, TUIPlaySource videoSource) {
        if (index >= mShortVideoView.getCurrentDataCount() - 1) {
            mShortViewRefresh.setRefreshing(true);
            ShortVideoModel.getInstance().loadMore(false);
        }
    }
});
```

4. 设置视频播放策略

设置视频播放过程中的各种策略。

策略 TUIPlayerVodStrategy 参数

需要使用 Builder 构建。

函数	描述
setPreloadCount	设置预加载最大并发数量，默认3。
setPreDownloadSize	设置预下载缓存大小，默认1MB，单位 MB。
setPreLoadBufferSize	设置预播放缓存大小，默认0.5MB，单位 MB。
setMaxBufferSize	设置播放时视频缓存大小，默认10MB，单位 MB。
setPreferredResolution	设置视频播放的偏好分辨率，默认720 x 1280。
setProgressInterval	播放进度回调间隔，默认500毫秒，单位毫秒。
setRenderMode	渲染平铺模式，默认0。liteavPlayer 中，0代表全屏屏幕，1代表按照视频实际比例渲染，可能会有黑边。
setExtInfo	设置额外信息。
setMediaType	当提前知道播放的媒资类型时，可以通过该接口设置媒资类型，减少播放器SDK内部播放类型探测，提升启播速度。
enableAutoBitrate	设置是否启用码率自适应。
setResumeMode	设置续播模式，分为三种模式： TUIConstants.TUIResumeMode.NONE：不续播。 TUIConstants.TUIResumeMode.RESUME_LAST：续播最近一次播放。 TUIConstants.TUIResumeMode.RESUME_PLAYED：续播所有播放过的视频。
setDisplayViewFactory	设置自定义视频图层，可通过实现 IDisplayViewFactory 来自定义视频图层
setEnableAccurateSeek	设置是否开启精准 seek，开启精准seek之后，seek 的精准度会大幅提升，但是 seek 会有耗时，关闭之后，实际的 seek 时间可能跟预期的有差距，差距根据视频的关键帧分布而定，但是 seek 耗时会变短。
setAudioNormalization	设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。注意：播放器高级版 11.7 版本开始支持。 可填预设值，相关常量类 TXVodConstants： • 关：AUDIO_NORMALIZATION_OFF • 开：AUDIO_NORMALIZATION_STANDARD（标准） • AUDIO_NORMALIZATION_LOW（低） • AUDIO_NORMALIZATION_HIGH（高）
setIsRetainPreVod	是否保留上一个播放器，以加快上一个播放器的起播速度。

setSuperResolution Mode	是否开启超分，相关常量： 关闭超分：TUIConstants.TUISuperResolution.SUPER_RESOLUTION_NONE 开启超分：TUIConstants.TUISuperResolution.SUPER_RESOLUTION_ASR 使用该能力需要集成超分插件和超分资源，具体可参见 超分文档 。 同时需要需要调用 TUI 配置超分资源目录： `TUIPlayerCore.setSuperResolutionResource(getApplicationContext(), destPath)`
setRetryCount	设置弱网下的点播重试次数，设置为 0 则不进行重试。

5. 填充数据

往 TUIShortVideoView 中填充数据：

```
mSuperShortVideoView.setModels(shortVideoBeanList);
```

填充数据的时候，指定初始位置：

```
mSuperShortVideoView.setModels(shortVideoBeanList, index);
```

追加数据：

```
mSuperShortVideoView.appendModels(shortVideoBeanList);
```

TUIVideoSource 类

参数/函数	参数类型	描述
setVideoURL	String	视频链接，建议填充该字段，会加快预加载速度。
setCoverPictureUrl	String	视频封面，会回调到 layer，由客户自行处理。
setAppId	int	视频 appId。
setFileId	String	视频 fileId。
setPSign	String	视频加密 pSign。
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。
setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的 layer 进行消息通知。该方法必须通过 TUIDataManager 获取到之后的 Source 调用才会有效。
setVideoConfig	TUIPlayerVideo Config	视频独立配置。
setExternalSubtitle	List<TUISubtitle Source>	设置外挂字幕，会自行加载到点播播放器中，必须要播放器高级版支持。

setAutoPlay	boolean	设置该视频是否自动播放。

TUIPlayerVideoConfig 类

参数	类型	描述
setPreloadBufferSizeInMB	float	设置视频单独的预播放缓存大小，可选。
setPreferredResolution	long	设置视频单独的起播和预加载分辨率，可选。
setPreDownloadSize	float	设置视频单独的预下载缓存大小，可选。

TUILiveSource 类

参数/函数	参数类型	描述
setUrl	String	直播链接。
setCoverPictureUrl	String	封面，会回调到 layer，由客户自行处理。
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。
setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的 layer 进行消息通知。该方法必须通过 TUIDataManager 获取到之后的 Source 调用才会有效。
setLiveConfig	TUIPlayerLiveConfig	独立配置。
setAutoPlay	boolean	设置是否自动播放，直播设置该字段为 false 时，一开始会没有画面，需要使用封面图遮盖。

TUIPlayerLiveConfig 类

参数	类型	描述
setCacheMinTime	float	缓存自动调整的最小时间，取值需要大于0。 **默认值**：1。
setCacheMaxTime	float	缓存自动调整的最大时间，取值需要大于0。 **默认值**：5

TUIPlaySource 类

参数/函数	参数类型	描述
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。

setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的layer进行消息通知。该方法必须通过 TUIDataManager 获取到之后的Source调用才会有效。

6. 操作列表数据

TUI短视频提供了数据操作接口，可以实时修改列表内已经添加的数据，通过 `TUIShortVideoView` 的 `getDataManager()` 方法获取数据操作对象，如下所示：

```
// 获取数据操作对象
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
```

获得数据操作对象之后，可以实时对列表数据进行操作，接口如下：

函数名称	返回参数	传入参数	描述
removeData	void	index：需要移除的页面位置。	移除对应的页面和数据。
removeRangeData	void	index：需要移除页面的起始位置。 count：从起始位置开始要移除的数量，不包括 count 的最后一位。	移除一个范围的页面和数据。
removeDataByIndex	void	removeIndexList：需要移除的页面的位置集合。	按照传入的索引，移除索引集合内所有的页面和数据。
addData	void	source：需要插入的数据。 index：插入数据的位置。	根据插入数据的位置，将传入的数据插入到指定位置。
addRangeData	void	sources：需要插入的数据集合。 startIndex：插入数据的起始位置。	根据传入的起始位置，将数据集合插入到指定位置。
replaceData	void	source：需要替换的数据。 index：需要被替换的位置。	根据传入的位置，把指定位置的数据替换为传入的数据。
replaceRangeData	void	sources：需要替换的数据集合。 startIndex：替换的起始位置。	根据传入的起始位置，将传入的数据集合替换到指定位置。
getDataByPageIndex	TUIVideoSource	index：页面位置。	根据传入的位置，获得指定位置的页面数据。
getCurrentDataCount	int	–	获得当前列表所有数据的总数。
getCurrentIndex	int	–	获得当前正在显示页面的位置。

getCurrentModel	TUIVideoSource	–	获得当前正在显示页面的数据。

7. 获取当前正在播放的视频资源

获取当前正在播放的视频资源，使用参见如下代码：

```
mSuperShortVideoView.getCurrentModel()
```

8. 暂停

暂停当前正在播放视频。

```
mSuperShortVideoView.pause();

// 所有视频起播后都会暂停，直到调用了 resume 来中断该状态
mSuperShortVideoView.pause(true)
```

9. 从指定位置播放

从指定位置开始播放，该方法可以在播放过程中直接跳转到指定位置，默认不平滑跳转，使用参见如下代码：

```
// index 为指定的页面位置
mSuperShortVideoView.startPlayIndex(index);


// index 为需要前往的位置，true 为需要平滑切换，默认为false
mSuperShortVideoView.startPlayIndex(index, true);
```

10. 设置短视频播放模式

目前短视频播放模式有两种，分别为列表循环播放 `TUIVideoConst.ListPlayMode.MODE_LIST_LOOP` ，当前视频播放完毕之后会自动播放下一个，播放到最后一个之后，会自动回到首个视频继续播放。以及单个视频循环播放 `TUIVideoConst.ListPlayMode.MODE_ONE_LOOP` ，会一直重复播放当前视频直到用户手动去滑动翻页，也可以设置为 `TUIVideoConst.ListPlayMode.MODE_CUSTOM` 由业务接管播放逻辑。当不设置的时候，默认为 `MODE_ONE_LOOP` ，设置方式如下：

```
// set to MODE_ONE_LOOP
mSuperShortVideoView.setPlayMode(TUIVideoConst.ListPlayMode.MODE_ONE_LOOP);
```

11. 销毁控件

销毁控件和资源。

```
mSuperShortVideoView.release()
```

12. 续播当前视频

续播当前视频，使用参见如下代码：

```
mSuperShortVideoView.resume()
```

13. 实时切换分辨率

TUI 短视频可以实时切换当前视频分辨率以及全局视频分辨率，接口如下：

```
mSuperShortVideoView.switchResolution(720 * 1080,
TUIConstants.TUIResolutionType.CURRENT);
```

switchType 除了可以传递 `TUIResolutionType` 以外，也可以直接指定需要切换的视频的 index 来切换视频分辨率。
switchType 含义如下：

参数	描述
GLOBAL	设置全局分辨率
CURRENT	设置当前视频分辨率
其他大于等于0的值	设置指定位置的分辨率

目前分辨率的优先级，当前视频分辨率的设置优先级大于全局分辨率。

14. 暂停和继续预加载

TUI 短视频可以实时暂停和继续预加载任务

```
// pause all preload
mSuperShortVideoView.pausePreload();
// start preload from current video index
mSuperShortVideoView.resumePreload();
```

调用继续预加载的时候，会从当前视频开始往后继续预加载。

15. 添加图层

TUI 短视频可以通过添加图层来实现短视频播放界面上的自定义 UI。当 onCreateItemLayer 回调时，就可以通过方法携带的 LayerManager 进行图层的添加和管理。可以通过继承 TUIBaseLayer 来自定义自己需要的图层。图层会浮在视频 VideoView 的上方。
图层的显示和隐藏，都是通过对 View 的添加和移除来操作的，不会产生过度渲染的问题。

onCreateVodLayer、onCreateLiveLayer和onCreateCustomLayer 参数

参数	类型	描述
layerManager	TUIVodLayerManager、	图层管理对象

	TUILiveLayerManager和TUICustomLayerManager	
viewType	int	当前视频播放类型 • TUIVideoConst.ItemType.ITEM_TYPE_VOD：点播 • TUIVideoConst.ItemType.ITEM_TYPE_LIVE：直播 • TUIVideoConst.ItemType.ITEM_TYPE_CUSTOM：自定义界面 • 其他由PlayerSource传入的自定义类型

```
layerManager.addLayer(new TUICoverLayer());
layerManager.addLayer(new TUIVideoInfoLayer());
layerManager.addLayer(new TUILoadingLayer());
layerManager.addLayer(new TUIErrorLayer());
```

16. 移除指定图层

将图层管理器中的图层移除。

```
layerManager.removeLayer(layer);
```

17. 移除所有图层

移除图层管理器中的所有图层。

```
layerManager.removeAllLayer();
```

18. 获得图层的当前层级

传入图层，获得当前图层的层级，依此可以判断图层显示过程中的先后覆盖顺序，以及触摸事件的先后传递顺序。

```
layerManager.indexOfLayer(layer);
```

19. 自定义列表滑行速率

TUI短视频提供了自定义列表滑行速率的接口，可通过 `TUIShortVideoView` 的 `setPageScroller` 方法来实现，示例如下：

```
mSuperShortVideoView.setPageScroller(new LinearSmoothScroller(getContext()){
    @Override
    protected float calculateSpeedPerPixel(DisplayMetrics displayMetrics) {
        // 返回滑行每一个像素所用的时间，单位：毫秒
        return 25f/displayMetrics.densityDpi;
    }
});
```

20. 禁止列表滑动

可以通过 `TUIShortVideoView` 的 `setUserInputEnabled` 方法来禁止或者允许列表滑动。示例如下：

```
// false 禁止用户滑动
mSuperShortVideoView.setUserInputEnabled(false);
```

21. 图层回调

Layer回调包括基类的 `ITUILayer` 回调、播放器事件通知，以及来自videoView的组件事件，继承 `TUIVodLayer` 、 `TUILiveLayer` 和 `TUICustomLayer` 后，可以根据功能需要来接收视频播放的回调，目前回调函数如下：

TUIVodLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
getVideoView	TUIBaseVideoView	-	获得当前 VideoView，如果 layerManager 与 VideoView 还未绑定会返回空。
getPlayerController	TUIPlayerController	-	获得当前播放 Controller，如果还未与 Controller 发生绑定，会返回空。
getPlayer	ITUIVodPlayer	-	获得当前播放器，如果还未与 Controller 发生绑定，会返回空。
getRenderMode	int	-	获得当前播放器画面渲染填充模式。
onControllerBind	void	controller：当前视频播放控制器	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在播放的视频。
onControllerUnBind	void	controller：当前视频播放控制器	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。

onBindData	void	videoSource：视频数据	当前 VideoView 绑定了视频数据。
onViewRecycled	void	videoView: 当前播放器 view 容器	当前 videoView 被回收。
onExtInfoChanged	void	videoSource：变化后的视频源	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通知。
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 layer 中，供 layer 释放资源。

TUILiveLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
getVideoView	TUIBaseVideoView	-	获得当前 VideoView，如果 layerManager 与 VideoView 还未绑定会返回空。
getPlayerController	TUIPlayerController	-	获得当前播放 Controller，如果还未与 Controller 发生绑定，会返回空。
getPlayer	ITUILivePlayer	-	获得当前播放器，如果还未与 Controller 发生绑定，会返回空。
getRenderMode	int	-	获得当前播放器画面渲染填充模式。
onControllerBind	void	controller：当前视频播放控制器	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在播放的视频。
onControllerUnBind	void	controller：当前视频播放控制器	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。

onBindData	void	videoSource：直播数据	当前 VideoView 绑定了视频数据。
onViewRecycled	void	videoView: 当前播放器 view 容器	当前 videoView 被回收。
onExtInfoChanged	void	source: 变化后的数据	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通知
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 layer 中，供 layer 释放资源

TUICustomLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
onControllerBind	void	-	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在显示。
onControllerUnBind	void	-	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。
onBindData	void	videoSource：数据	当前 VideoView 绑定了视频数据。
onViewRecycled	void	videoView: 当前view 容器	当前 videoView 被回收。
onExtInfoChanged	void	source: 变化后的数据	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通知
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 layer 中，供 layer 释放资源

TUIVodObserver 类

函数	返回类型	参数	描述
onPlayPrepare	void	-	视频准备完毕
onPlayBegin	void	-	视频开始播放
onPlayPause	void	-	视频暂停
onPlayStop	void	-	视频停止
onPlayLoading	void	-	视频开始加载
onPlayLoadingEnd	void	-	视频加载结束
onPlayProgress	void	• current：当前视频播放进度，单位毫秒，long 类型。 • duration：当前视频总时长，单位毫秒，long 类型。 • playable：当前视频可播放时长，单位毫秒，long 类型	视频播放进度
onSeek	void	position：跳转到的视频进度。单位秒，int 类型。	视频进度发生跳转
onError	void	code：视频错误码 message：错误描述	视频播放发生错误
onRcvFirstIframe	void	-	收到首帧事件
onRcvAudioTrackInformation	void	infoList：音频轨道	收到音频轨道信息
onRcvSubTitleTrackInformation	void	infoList：视频字幕信息	收到视频字幕信息
onRecFileVideoInfo	void	params：视频文件信息	收到视频文件信息，一般只使用 fileId 播放才会触发该回调。
onResolutionChanged	void	width：视频宽度 height：视频高度	当前视频分辨率发生变化
onPlayEvent	void	player：播放器 event：事件id bundle：事件内容	播放器所有事件通知
onPlayEnd	void	-	当前视频播放结束

TUILiveObserver 类

函数	参数	描述

onError	• player: 当前直播播放器 • code：错误码 • msg：错误信息 • extraInfo：附加消息	播放发生错误
onWarning	• player: 当前直播播放器 • code：错误码 • msg：警告信息 • extraInfo：附加消息	播放器发生警告
onVideoResolutionChanged	• player: 当前直播播放器 • width: 视频宽度 • height: 视频高度	播放器分辨率发生变化
onConnected	• player：当前直播播放器 • extraInfo：附加信息	数据流连接成功
onVideoPlaying	• player：当前直播播放器 • firstPlay：是否是首次播放，**可以用来判断首帧。** • extraInfo：附加信息	视频开始播放
onAudioPlaying	• player：当前直播播放器 • firstPlay：是否是首次播放 • extraInfo：附加信息	音频开始播放
onVideoLoading	• player：当前直播播放器 • extraInfo：附加信息	视频开始加载
onAudioLoading	• player：当前直播播放器 • extraInfo：附加信息	音频开始加载
onPlayoutVolumeUpdate	• player：当前直播播放器 • volume：变化后的音量	播放器音量大小回调
onStatisticsUpdate	• player：当前直播播放器 • statistics：数据详情	直播播放器统计数据回调
onSnapshotComplete	• player：当前直播播放器 • image：截图的图片	截图回调
onRenderVideoFrame	• player：当前直播播放器 • videoFrame：图像帧	自定义视频渲染回调，调用enableObserveVideoFrame后开启回调。
onPlayoutAudioFrame	• player：当前直播播放器 • audioFrame：音频帧	自定义音频数据回调，调用enableObserveAudioFrame开启回调。

onReceiveSeiMessage	• player：当前直播播放器 • payloadType：回调数据的SEI payloadType • data：数据	收到 SEI 消息的回调，发送端通过V2TXLivePusher中的sendSeiMessage来发送 SEI 消息。

onStreamSwitched	• player：当前直播播放器
	• url：切换的url
	• code：状态码，
	  ○ 0：成功
	  ○ -1：切换超时
	  ○ -2：切换失败，服务端错误。
	  ○ -3：切换失败，客户端错误。
		分辨率无缝切换回调

onLocalRecordBegin	• player: 当前直播播放器
	• code: 状态码。
	  ○ 0：录制任务启动成功。
	  ○ -1：内部错误导致录制任务启动失败。
	  ○ -2：文件后缀名有误（例如不支持的录制格式）。
	  ○ -6：录制已经启动，需要先停止录制。
	  ○ -7：录制文件已存在，需要先删除文件。
	  ○ -8：录制目录无写入权限，请检查目录权限问题。
	• storagePath: 录制的文件地址。
		录制任务开始的事件回调

onLocalRecording	• player：当前直播播放器 • durationMs：录制时长 • storagePath：录制的文件地址。	录制任务正在进行中的进展事件回调

onLocalRecordComplete	• player：当前直播播放器
	• code：状态码。
	  ○ 0：结束录制任务成功。
	  ○ -1：录制失败。
	  ○ -2：切换分辨率或横竖屏导致录制结束。
	  ○ -3：录制时间太短，或未采集到任何视频或音频数据，请检查录制时长，或是否已开启音、视频采集。
	• storagePath: 录制的文件地址。
		录制任务已经结束的事件回调

22. 播放器函数

在layer中，可以通过 `getPlayer` 获得播放器对象，播放器 ITUIVodPlayer 和 ITUILivePlayer 的接口函数如下：

ITUIVodPlayer 类

函数	返回类型	参数	描述
prePlay	void	model：视频数据，类型 TUIVideoSource	对视频进行预播放，内部会有判重机制。
resumePlay	void	-	继续播放当前视频。
seekTo	void	time：需要跳转的时间点，单位秒，int 类型。	跳转到指定播放位置。
isPlaying	boolean	-	当前视频是否正在播放。
startPlay	void	model：视频数据，类型 TUIVideoSource。	播放视频
pause	void	-	暂停播放
stop	void	needClearLastImg：可选参数，是否清除当前画面	停止播放
getCurrentPlayTime	float	-	获得当前播放时间，单位：秒。
setDisplayView	void	videoView：视频渲染 View，类型 TUIVideoRenderView。	设置播放器要渲染的 View。
setSurface	void	surface：画布	设置视频要渲染的 surface。
addPlayerObserver	void	observer：播放器监听，类型 PlayerObserver。	设置播放器监听。
removePlayerObserver	void	observer：播放器监听，类型 PlayerObserver。	移除播放器监听。
setConfig	void	config：视频配置，类型 TXVodPlayConfig。	设置视频配置。
setRenderMode	void	renderMode：渲染模式， FULL_FILL_SCREEN：保持宽高比且铺满 ADJUST_RESOLUTION: 保持宽高比展示视频	设置平铺模式。
setStartTime	void	startTime：开始播放时间，单位：秒。float 类型	设置开始播放时间，在播放前调用有效，只会生效一次。循环播放后会从头开始。
setLoop	void	isLoop：是否循环	设置视频是否循环播放。
setBitrateIndex	void	index：码率角标	设置当前码率。

switchResolution	void	resolution：分辨率，即宽 × 高	设置改播放器的分辨率，如果有对应分辨率就会进行切换，仅对本次播放生效。
getSupportResolution	List<TUIPlayerBitrateItem>	–	获取当前正在播放视频的分辨率列表。
getBitrateIndex	int	–	获取当前正在播放视频的码率角标。
setRate	void	rate：视频速率，正常速率为1.0	设置当前播放器播放速率。
getCurrentPlaybackTime	float	–	获取当前播放器已播放时间，单位 秒。
getBufferDuration	float	–	获取当前视频已经缓冲的时间，单位 秒。
getDuration	float	–	获取当前视频的总时长，单位 秒。
getPlayableDuration	float	–	获取当前视频可播放的事件，单位 秒。
getWidth	int	–	获取当前正在播放视频的宽度。
getHeight	int	–	获取当前正在播放视频的高度。
setRenderRotation	void	rotation：角度	设置当前视频旋转角度。
enableHardwareDecode	boolean	enable：是否开启硬解	设置当前播放器是否开启硬解。
setMute	void	mute：是否静音	设置当前播放视频是否静音。
setAudioPlayoutVolume	void	volume：视频音量，0到100	设置当前视频的输出音量。
setRequestAudioFocus	boolean	requestFocus：是否获取音频焦点	设置当前播放器是否获取音频焦点。
snapshot	void	listener：TXLivePlayer.ITXSnapshotListener类型，截图回调	对当前播放器正在播放的视频进行截图。
setMirror	void	mirror：是否镜像	设置当前视频是否镜像播放。
setToken	void	token：加密 HLS 的 token	设置加密 HLS 的 token。

版权所有：腾讯云计算（北京）有限责任公司

isLoop	boolean	-	当前播放器是否是循环播放。
attachTRTC	void	trtcCloud：trtc服务对象	把播放器的音视频流通过 TRTC 进行推送，更多 TRTC 服务请参见 TRTC 产品概述 。
detachTRTC	void	-	点播解绑 TRTC 服务。
setStringOption	void	• key：业务参数键值 • value：业务参数值	设置播放器业务参数。
setSubtitleView	void	subtitleView：字幕组件	设置视频字幕的组件。
addSubtitleSource	void	• url：字幕链接 • name：字幕名称 • mimeType：字幕格式	向视频添加字幕源。
selectTrack	void	trackIndex：音视频轨道	添加选择音视频轨道，目前常用于多音轨、字幕的选择。
deselectTrack	void	trackIndex：音视频轨道	移除选择音视频轨道。
setAudioNormalization	void	value：响度范围：-70~0 (LUFS)，同时支持自定义数值。注意：播放器高级版 11.7 版本开始支持。可填预设值，相关常量 TXVodConstants： • 关：AUDIO_NORMALIZATION_OFF • 开：AUDIO_NORMALIZATION_STANDARD（标准） • AUDIO_NORMALIZATION_LOW（低） • AUDIO_NORMALIZATION_HIGH（高）	设置音量均衡。
setSubtitleStyle	void	renderModel：字幕格式。	设置字幕格式。
getSubtitleTrackInfo	List<TXTrackInfo>	-	获取导入的字幕轨道信息，prepare 事件之后调用才有效。
getAudioTrackInfo	List<TXTrackInfo>	-	获取导入的音频轨道信息，prepare 事件之后调用才有效。
setVodSubtitleDataListener	void	listener:自定义字幕回调	设置自定义字幕回调，播放器会将字幕文本回调出来，使用该能力需要提前在 TUIVodSource 的 config 中设置参数：

			TUIPlayerVideoConfig config = new TUIPlayerVideoConfig(); Map<String, Object> map = new HashMap<>(); map.put("450", 0); config.setExtInfo(map); source.setVideoConfig(config);

TUIPlayerBitrateItem 类

函数	返回类型	描述
getIndex	int	当前分辨率的码率角标
getWidth	int	当前分辨率的视频宽度
getHeight	int	当前分辨率的视频高度
getBitrate	int	当前分辨率的视频码率

ITUILivePlayer 类

函数	返回类型	参数	描述
prePlay	void	model：视频数据，类型 TUILiveSource	对直播进行预播放，内部会有判重机制。
resumePlay	void	-	继续播放当前视频。
setConfig	void	config：直播配置	设置播放器配置
addLiveObserver	void	observer: 直播事件观察者	订阅直播事件
removeLiveObserver	void	observer: 直播事件观察者	移除直播事件订阅
switchStream	int	model：需要切换的数据源	切换当前直播源的其他分辨率流，返回0代表成功
setRenderRotation	int	rotation：旋转方向	设置当前画面旋转方向
setPlayoutVolume	int	volume：音量大小，取值范围0 - 100。**默认值:** 100	设置当前音量大小
getStreamList	ArrayList<V2TXLiveDef.V2TXLiveStreamInfo>	-	获取当前码流信息

enableVolumeEvaluation	int	value：决定了 onPlayoutVolumeUpdate 回调的触发间隔，单位为ms，最小间隔为100ms，如果小于等于0则会关闭回调，建议设置为300ms。 **默认值**：0，不开启。	启用播放音量大小提示
snapshot	int	–	截取播放过程中的视频画面。 返回值： • V2TXLiveCode。 • V2TXLIVE_OK：成功。 • V2TXLIVE_ERROR_REFUSED：播放器处于停止状态，不允许调用截图操作。

enableReceiveSeiMessage	int	• enable：**默认值**: false。
		  ○ true：开启接收 SEI 消息。
		  ○ false：关闭接收 SEI 消息。
		• payloadType：指定接收 SEI 消息的 payloadType，支持 5、242，请与发送端的 payloadType 保持一致。
			开启接收 SEI 消息

showDebugView	void	isShow：是否显示。**默认值**: false。	是否显示播放器状态信息的调试浮层
setProperty	int	key：高级 API 对应的 key。 value：调用 key 所对应的高级 API 时，需要的参数。	调用 V2TXLivePlayer 的高级 API 接口
startLocalRecording	int	params:本地录制音视频配置	开始录制音视频流。 返回值 V2TXLiveCode。 • V2TXLIVE_OK：成功。 • V2TXLIVE_ERROR_INVALID_PARAMETER：参数不合法，例如 filePath 为空。 • V2TXLIVE_ERROR_REFUSED：API被拒绝，拉流尚未开始。
stopLocalRecording	void	–	停止录制音视频流。当停止拉流后，如果视频还在录制中，SDK 内部会自动结束录制。

23. 设置列表动画

TUI 短视频列表支持设置元素在被添加、移除等UI 变动时候的动画效果，示例如下：

```
mShortVideoView.setItemAnimator(new DefaultItemAnimator());
```

其中 ItemAnimator 与 RecyclerView 的 ItemAnimator 一致。

24. 全局监听视频预下载变动

TUI Core 提供了静态接口，可全局监听视频预下载的状态变化，示例如下：

```
TUIPreloadApi.getInstance().addObserver(new TUIPreloadObserver() {
    @Override
    public void onVodPreloadStatusChanged(TUIVideoSource vodSource, int state, long
resolution) {

    }
});
```

其中 state 状态可参考 `TUIConstants#TUIPreLoadState` 。

高级功能

1. 业务通知消息到页面图层

TUI 提供了消息接口供用户把数据实时通知到当前图层，可以通过数据操作对象获取到视频对象之后，进行通知，示例如下：

```
// get data controller
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
// get dataSource
TUIPlaySource videoSource = dataManager.getDataByPageIndex(0);
Map<String, String> data = new HashMap<>();
data.put("key1", "data1");
// set extInfo and notify to layer
videoSource.setExtInfoAndNotify(data);
```

随后在 layer 的 `onExtInfoChanged` 回调中会收到该通知，从而对当前页面进行 UI 修改，示例如下：

```
@Override
public void onExtInfoChanged(TUIVideoSource videoSource) {
    super.onExtInfoChanged(videoSource);
    Map<String, String> data = (Map<String, String>) videoSource.extInfo;
    Log.i("tag", "get data:" + data);
}
```

> **注意：**
> `onExtInfoChanged` 只会在 `onBindData` 绑定数据之后才会触发。

2. 点播设置音量均衡

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

> **注意：**
> 播放器高级版 11.7 版本开始支持。

```
/*
 * 支持自定义数值，响度范围：-70～0(LUFS)
 * 预设值  关：TXVodConstants#AUDIO_NORMALIZATION_OFF
 *         开（低）：TXVodConstants#AUDIO_NORMALIZATION_LOW
 *         开（标准）：TXVodConstants#AUDIO_NORMALIZATION_STANDARD
 *         开（高）：TXVodConstants#AUDIO_NORMALIZATION_HIGH
 */
TUIPlayerVodStrategy vodStrategy = new TUIPlayerVodStrategy.Builder()
        .setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD)
        .build();
mShortVideoView.setVodStrategy(vodStrategy);
```

3. 点播终端极速高清

TUI 短视频支持开启终端高清，终端极速高清介绍可参见 超分文档 。
根据文档中的步骤，将超分插件下载下来，并依赖到项目中。再根据文档中的提示，下载安卓超分资源，解压并复制到 app 的 assets 目录中。随后调用 TUIPlayerCore.setSuperResolutionResource 设置业务侧在设备上的资源缓存目录文件夹，TUI 会在 assets 中寻找超分资源 sr_resource 文件夹，自行复制到您设置到目录中，并将资源在播放视频的时候进行加载。

```
String destPath = getCacheDir().getAbsolutePath() + "/sr_resource";
File file = new File(destPath);
if (!file.exists()) {
    file.mkdir();
}
TUIPlayerCore.setSuperResolutionResource(getApplicationContext(), destPath);
```

随后在设置点播策略的时候，开启超分策略。

```
TUIPlayerVodStrategy vodStrategy = new TUIPlayerVodStrategy.Builder()

.setSuperResolutionMode(TUIConstants.TUISuperResolution.SUPER_RESOLUTION_ASR)
        .build();
mShortVideoView.setVodStrategy(vodStrategy);
```

4. 自定义下拉刷新和上拉加载能力

TUIShortVideoView 暴露了 computeVerticalScrollOffset、computeVerticalScrollRange 和 computeVerticalScrollExtent 三个方法用于计算当前列表是否滑动到最顶部、或者滑动最底部。其中：

`computeVerticalScrollOffset` ：当前垂直滚动的偏移量（像素）。
`computeVerticalScrollRange` ：RecyclerView 的垂直滚动范围（总高度，像素）。
`computeVerticalScrollExtent` ：当前滚动视图中可见项的总高度（像素）。
当滑动过程中，判断 computeVerticalScrollOffset 方法如果小于等于 0，则代表滑动到最顶部，此时可将触摸事件传递给外层布局来进行下拉刷新。
滑动过程中，可以使用当前界面可见高度 computeVerticalScrollExtent 加上已经划过的距离 computeVerticalScrollOffset 之和，如果大于等于 computeVerticalScrollRange 总高度来判断是否滑动到最底部，代码如下：

```
  int verticalScrollOffset = recyclerView.computeVerticalScrollOffset();
  int verticalScrollRange = recyclerView.computeVerticalScrollRange();
  int verticalScrollExtent = recyclerView.computeVerticalScrollExtent();

  boolean isAtBottom = verticalScrollOffset + verticalScrollExtent >=
verticalScrollRange;
```

iOS

最近更新时间：2025-03-11 16:52:02

组件简介

TUIPlayerShortVideo 组件是腾讯云推出的一款性能优异，支持视频极速首帧和流畅滑动，提供优质播放体验的短视频组件。

- 首帧秒开：首帧时间是短视频类应用核心指标之一，直接影响用户的观看体验。短视频组件通过预播放、预下载、播放器复用和精准流量控制等技术，实现极速首帧、滑动丝滑的优质播放体验，从而提升用户播放量和停留时长。
- 优异的性能：通过播放器复用和加载策略的优化，在保证极佳流畅度的同时，始终让内存和 CPU 消耗保持在较低的水平。
- 快速集成：组件对复杂的播放操作进行了封装，提供默认的播放 UI，同时支持 FileId 和 URL 播放，可低成本快速集成到您的项目中。

效果对比

从下方示例视频您可以看到在接入短视频最佳策略前后的对比差异。

- 优化前有明显的首帧卡顿感。
- 优化后播放流畅丝滑，优化后起播平均时长达到10毫秒 – 30毫秒。

未优化短视频	优化后短视频
观看视频	观看视频

TUIPlayerKit 下载

TUIPlayerKit SDK 和 Demo 可 单击这里 下载。

集成指引

1. 依赖库

TUIPlayerShortVideo 依赖的 SDK 有：

- TUIPlayerCore
- TXLiteAVSDK ≥ 11.4
- SDWebImage
- Masonry

2. 环境要求

- 系统版本：≥ iOS 9.0
- 开发环境：≥ Xcode 14.0（推荐使用最新版本）

3. 集成 TUIPlayerCore

解压下载的 TUIPlayerKit 资源包，将 TUIPlayerCore.xcframework 组件 SDK 添加到您项 中 Xcode Project 的合适位置并选择合适的 target，同时勾选 Do Not Embed 。

4. 集成 TUIPlayerShortVideo

解压下载的 TUIPlayerKit 资源包，将 TUIPlayerShortVideo.xcframework 组件 SDK 添加到您项 中 Xcode Project 的合适位置并选择合适的 target，同时勾选 Do Not Embed 。

5. 集成 TXLiteAVSDK

TXLiteAVSDK 集成方法请参见 TXLiteAVSDK 集成指引 。

6. 集成 SDWebImage

SDWebImage 的下载和集成请参见 GitHub 指引 。

7. 集成 Masonry

Masonry 的下载和集成请参见 GitHub 指引 。

8. Pod 方式集成

如果您的项目支持 pod，那么也可以通过我们提供的 spec 文件集成，如下：

```
pod 'TUIPlayerShortVideo' ,:path => '../../../SDK/TUIPlayerShortVideoSDK/'
pod 'TUIPlayerCore' ,:path => '../../../SDK/TUIPlayerCoreSDK/'
```

注意：

- Path 请根据自己的项目文件路径自行配置。
- 目前暂不支持远程 Pod 集成。

接口使用说明

1. 快速接入

1.1. 配置播放器高级版 Licence

使用 TUIPlayer Kit 组件需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引获取。若您已获取对应 License，可前往 腾讯云视立方控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请移动端播放器高级版 License，将会出现视频播放失败、黑屏等现象。
调用相关功能之前，在您的项目中配置 Licence。引用 TUIPlayerCore 模块建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中，做如下配置：

```
NSString * const licenceURL = @"<获取到的licenseUrl>";
NSString * const licenceKey = @"<获取到的key>";
[TXLiveBase setLicenceURL:licenceUrl key:licenceKey];
[[TXLiveBase sharedInstance] setDelegate:self];
```

1.2. 播放

初始化 TUIShortVideoView，举例如下：

```
- (TUIShortVideoView *)videoView {

    if (!_videoView) {
        ///设置自定义UI
        TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager
alloc] init];
        [uiManager setControlViewClass: TUIPlayerShortVideoControlView.class];
        [uiManager setControlViewClass: TUIPSControlLiveView.class
viewType:TUI_ITEM_VIEW_TYPE_LIVE];
        [uiManager setControlViewClass: TUIPSControlCustomView.class
viewType:TUI_ITEM_VIEW_TYPE_CUSTOM];
        [uiManager setLoadingView:[[TUIPSDLoadingView alloc] init]];

        _videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
        _videoView.delegate = self;
        _videoView.customCallbackDelegate = self;
        //_videoView.isAutoPlay = NO;

        // Set your playback strategy
        TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
        model.mPreloadConcurrentCount = 1;
        model.preDownloadSize = 1;
        model.enableAutoBitrate = NO;
        model.isLastPrePlay = YES;

        // live strategy
        TUIPlayerLiveStrategyModel *liveStrategyModel = [[TUIPlayerLiveStrategyModel
alloc] init];
        liveStrategyModel.isLastPrePlay = YES;
        [_videoView setShortVideoLiveStrategyModel:liveStrategyModel];
    }
    return _videoView;
}
```

将 TUIShortVideoView 的实例添加到您想呈现的 View 上，参见如下代码：

```
videoView.frame = self.view.bounds;
[self.view addSubview:self.videoView];
```

然后添加您的视频数组：

```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///视频数据
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///直播数据
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///自定义数据
/// 这里根据您的业务情况，自行决定每一页的数据量
NSArray *videos1 = @[model1,model2,model3];
```

```
[self.videoView setShortVideoModels:videos1];
```

第一组视频播放完之后您还需要在 TUIShortVideoViewDelegate 的代理方法内继续拼入您的第二组视频数据：

```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///视频数据
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///直播数据
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///自定义数据
/// 这里根据您的业务情况，自行决定每一页的数据量
NSArray *videos2 = @[model1,model2,model3];
-(void)onReachLast {
    ///这里您可以做数据index索引记录，继续拼入您的第 3 4 5 6....组数据
    [self.videoView appendShortVideoModels:videos2];
}
```

1.3. TUIShortVideoView

TUIShortVideoView 主要接口如下：

参数名称	含义
isAutoPlay	首次加载是否自动播放第一个视频，默认 YES。
videos	只读属性，获取当前存在于视频列表中的数据。
currentVideoModel	当前正在播放的视频模型。
currentVideoIndex	当前正在播放的视频索引。
currentPlayerStatus	当前播放器的播放状态。
isPlaying	当前播放器是否正在播放。
delegate	代理。
refreshControl	设置下拉刷新控件。
initWithUIManager	初始化（带自定义 UI）。
setShortVideoStrategyModel	设置直播播放策略。
setShortVideoLiveStrategyModel	设置直播播放策略。
setShortVideoModels	首次设置数据源。
appendShortVideoModels	追加视频数据源。
removeAllVideoModels	删除所有视频数据。
setPlaymode	视频播放模式，单个循环或列表循环，默认前者。

pause	暂停。
resume	继续播放。
destoryPlayer	销毁播放器。
didScrollToCellWithIndex	跳到指定索引的视频。
startLoading	展示 loading 图。
stopLoading	隐藏 loading 图。
currentPlayerSupported Bitrates	当前正在播放的视频支持的码率。
bitrateIndex	获取当前正在播放的码率索引。
switchResolution:index:	切换分辨率。
pausePreload	暂停预加载。
resumePreload	恢复预加载。
getDataManager	获取数据管理器。
getVodStrategyManager	获取点播策略管理器。
getLiveStrategyManager	获取直播策略管理器。
setPlayRate:	设置当前播放器倍速（2.1.0.28版本新增）。

TUIShortVideoViewDelegate 主要接口如下：

参数名称	含义
scrollViewDidScrollContentOffset:	列表滑动中回调内容的偏移量。
scrollViewDidEndDeceleratingIndex: videoModel:	手动滑动停止回调当前索引和数据模型。
scrollToVideoIndex:videoModel:	手动/代码滑动回调将要播放的索引和数据模型。
onReachLast	需要加载下一页回调，回调时机跟 TUIPlayerVodStrategyModel/mPreloadConcurrentCount 相关。 例如：mPreloadConcurrentCount=3时，此方法会在视频组的倒数第三个回调提醒加载新的一页数据。
currentVideo:statusChanged:	视频播放器状态回调。
currentVideo:currentTime:totalTime:progress:	视频播放器进度回调。
onNetStatus:withParam:	视频播放器网络状态回调。

videoPreLoadStateWithModel:	视频预加载回调。
shortVideoView:willDefocusVideo:context:	即将反聚焦视频。
shortVideoView:didEndScrollingAnimationWithIndex:videoModel:	自动滚动停止回调（2.1.0.44版本新增）。
shortVideoView:playerDidReady:videoModel:	播放器 ready 回调（加载前），可做播放器配置（2.1.0.44版本新增）。
shortVideoView:widgetFrameForVideoModel:containerSize:	获取外部指定的播放器 widget 位置信息（2.1.0.49版本新增）。

2. 全局配置

您可以通过 TUIPlayerConfig 模型在 TUIPlayerCore 里设置一些全局配置。
TUIPlayerConfig 主要参数参见下表：

参数名称	含义
enableLog	是否允许打印日志，默认 NO。

然后通过 TUIPlayerCore 进行全局配置：

```
TUIPlayerConfig *config = [TUIPlayerConfig new];
config.enableLog = YES;
[[TUIPlayerCore shareInstance] setPlayerConfig:config];
```

3. 播放器策略配置

3.1. 点播播放策略设置

您可以通过 TUIPlayerVodStrategyModel 模型配置点播的播放策略。
TUIPlayerVodStrategyModel 主要参数参见下表：

参数名称	含义
mPreloadConcurrentCount	视频缓存个数，默认3。
mPreloadBufferSizeInMB	预播放大小，单位MB，默认0.5MB。
preferredResolution	偏好分辨率，默认720 * 1280。
progressInterval	进度条回调间隔时长，单位毫秒，默认500ms。
renderMode	画布填充样式，默认图像适应屏幕，保持画面完整。
extInfoMap	额外参数，预留。
enableAutoBitrate	是否开启自适应码率，默认 NO。

mediaType	设置媒资类型。
maxBufferSize	最大预加载大小，单位 MB ，默认10MB，此设置会影响 playableDuration，设置越大，提前缓存的越多。
mResumeModel	续播模式，默认 TUI_RESUM_MODEL_NONE。
preDownloadSize	预下载大小，单位 MB，默认1MB。
enableAccurateSeek	是否精确 seek，默认 YES。开启精确后 seek，seek 的时间平均多出 200ms。
audioNormalization	音量均衡响度范围：-70～0(LUFS)。此配置需要 LiteAVSDK 11.7 及以上版本支持。 以下几种常量供参考使用： • 关：AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h) • 开（标准响度）：AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h) • 开（低响度）：AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h) • 开（高响度）：AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h) 默认值为 AUDIO_NORMALIZATION_OFF。
isLastPrePlay	是否保留上一个预播放，默认 NO。
superResolutionType	超分类型，默认0。 **注意：** 开启超分需要先集成超分插件，否则无效。详情参见 终端极速高清 。
subtitleRenderModel	字幕样式。
headers	自定义 HTTP Headers。

然后进行播放器策略配置：

```
TUIPlayerStrategyModel *model = [[TUIPlayerStrategyModel alloc] init];
model.mPreloadConcurrentCount = 1;
model.preDownloadSize = 1;
model.enableAutoBitrate = NO;
model.mRenderMode = TUI_RENDER_MODE_FILL_SCREEN;
model.mResumeModel = TUI_RESUM_MODEL_LAST;
[_videoView setShortVideoStrategyModel:model];
```

3.2. 直播播放策略设置

您可以通过 TUIPlayerLiveStrategyModel 模型配置点播的播放策略。
TUIPlayerLiveStrategyModel 主要参数参见下表：

参数名称	含义
isLastPrePlay	是否保留上一个预播放，默认NO。
mRenderMode	画布填充样式，默认 V2TXLiveFillModeFill。
enablePictureInPicture	• YES：开启画中画功能。 • NO：关闭画中画功能。 **默认值**：NO。
volume	播放器音量，取值范围0 – 100。**默认值**：100。
maxAutoAdjustCacheTime	播放器缓存自动调整的最大时间，单位秒，取值需要大于0，默认值：5。
minAutoAdjustCacheTime	播放器缓存自动调整的最小时间，单位秒，取值需要大于0，默认值为1。
isShowDebugView	是否显示播放器状态信息的调试浮层，**默认值**：NO。

然后进行播放器策略配置：

```
TUIPlayerLiveStrategyModel *liveStrategyModel = [[TUIPlayerLiveStrategyModel alloc]
init];
[_videoView setShortVideoLiveStrategyModel:liveStrategyModel];
```

3.3. 动态策略调整

点播和直播的策略均支持动态的调整，步骤如下：

3.3.1. 通过 TUIShortVideoView 获取直播&点播策略管理类。

```
TUIPlayerVodStrategyManager *VodStrategyManager = [_videoView
getVodStrategyManager];
TUIPlayerVodStrategyManager *LiveStrategyManager = [_videoView
getLiveStrategyManager];
```

3.3.2. 通过 VodStrategyManager 和 LiveStrategyManager 调整播放策略。

```
[VodStrategyManager setRenderMode:TUI_RENDER_MODE_FILL_EDGE];
[LiveStrategyManager setRenderMode:V2TXLiveFillModeFill];
```

4. 数据管理

4.1. 数据模型

TUIShortVideoView 的原始数据模型包括：

参数说明	含义

TUIPlayerDataModel	基本的数据类型。
TUIPlayerVideoModel	点播数据类型，继承于 TUIPlayerDataModel。
TUIPlayerLiveModel	直播数据类型，继承于 TUIPlayerDataModel。

TUIPlayerDataModel

参数说明	含义
modelType	模型类型。
extInfo	点播数据类型。
onExtInfoChangedBlock	直播数据类型。
extInfoChangeNotify	通知 extInfo 数据发生改变。
asVodModel	强转为 TUIPlayerVideoModel 类型。
asLiveModel	强转为 TUIPlayerLiveModel 类型。

TUIPlayerVideoModel

参数说明	含义
videoUrl	视频 Url 地址。
coverPictureUrl	封面图。
duration	视频时长。
appId	appid
fileId	视频的 fileId。
pSign	签名字串。
subtitles	字幕信息。
config	视频的单独配置，详情请看 TUIPlayerVideoConfig。

TUIPlayerLiveModel

参数说明	含义
liveUrl	直播 Url
coverPictureUrl	封面图

4.2. 模型的构建

构建一组点播数据模型：

```
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.videoUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
model.duration = @"xxxx";
model.appId = @"xxxx";
model.fileId = @"xxxx";
model.pSign = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"titile":@"This is a vod broadcast interface",
    @"des":@"This is a vod broadcast interface"
};
model.extInfo = extr;
[modelArray addObject:model];
```

构建一组直播数据模型：

```
TUIPlayerLiveModel *model = [[TUIPlayerLiveModel alloc] init];
model.liveUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"liveTitile":@"This is a live broadcast interface",
    @"liveDes":@"This is a live broadcast interface"
};
model.extInfo = extr;
```

构建一组其他类型的数据模型：

```
/// 1 轮播图
TUIPlayerDataModel *model = [[TUIPlayerDataModel alloc] init];
NSDictionary *extr = @{
    @"images":@"xxxx",
    @"url":@"https://cloud.tencent.com",
    @"titile":@"This is a picture carousel display interface",
    @"des":@"This is a picture carousel display interface",
    @"name":@"@Mars",
    @"type":@"imageCycle"
};
model.extInfo = extr;
[modelArray insertObject:model atIndex:1];

/// 2 图文广告
TUIPlayerDataModel *model1 = [[TUIPlayerDataModel alloc] init];
```

```
NSDictionary *extr1 = @{
    @"adUrl":@"https://cloud.tencent.com",
    @"adUrl":@"https://cloud.tencent.com/document/product",
    @"adTitile":@"This is a web display interface",
    @"adDes":@"This is a web display interface",
    @"name":@"@Mars",
    @"type":@"ad"
};
model1.extInfo = extr1;
[modelArray insertObject:model1 atIndex:1];
```

注意：

- TUIPlayerDataModel 是一个大类，适用于所有非点播和非直播的数据类型。
- 用户可以通过 extInfo 业务字短继续做更细的分类，例如上面通过 TUIPlayerDataModel 分别构建出了“轮播图”和“图文广告”两种类型的数据，可通过 extInfo/type 去自行分类。
- extInfo 为一个灵活字短，用户可以随意去设计自己所需要的数据结构。

4.3. 数据的动态调整

TUIShortVideoView 提供了可供外部操作数据的数据管理类 TUIShortVideoDataManager，其作用主要是对当前播放器列表内的数据做基本增删改查等操作，参见如下演示代码：

```
///1、删除索引1处的数据和视图
[[self.videoView getDataManager] removeData:1];
///2、添加一组数据到索引9处
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.viewType = TUI_ITEM_VIEW_TYPE_CUSTOM;
[[self.videoView getDataManager]  addData:model index:9];
```

更详细的接口说明如下：

参数名称	含义
removeData	按索引移除数据。
removeRangeData	按范围移除数据。
removeDataByIndex	按索引数组移除数据。
addData:index	按索引添加数据。
addRangeData:startIndex	按模型数组从某个索引处添加数据。
replaceData:index	按索引替换数据。
replaceRangeData:startIndex	按模型数组从某个索引处替换数据。
getDataByPageIndex	读取某个索引处的数据。

getCurrentDataCount	获取当前播放列表内的数据总数。
getCurrentIndex	获取当前播放界面的数据索引。
getCurrentModel	获取当前播放界面的数据模型。

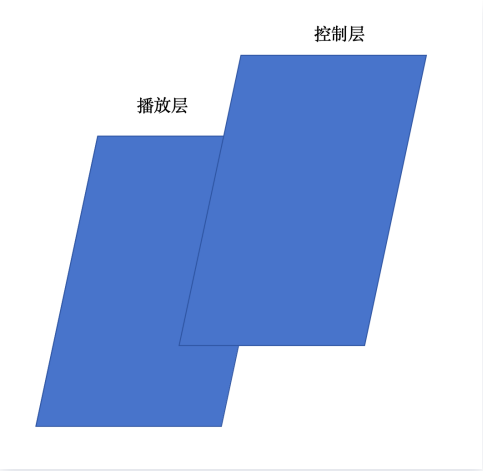
说明：

- DataManager 的接口调用完后 UI 界面自动刷新。
- 未操作当前播放界面的情况下，无感刷新。
- 操作了当前界面会刷新当前的界面。
- 如果删除了当前播放界面，会自动播放下一个，如果下一个无数据（已经到达末尾），将会播放上一个。

5. 自定义 UI 图层

5.1. 层级结构

TUIPlayerShortVideo 的层级结构大致如下：



分为显示层和控制层，两者是以堆叠的方式结合。

- **显示层负责内容展示，点播，直播，广告推广页面等，这层是 SDK 内部管理。**
- **控制层负责交互，点赞，评论，等，这层交给用户去实现高度的自定义。**

5.2. TUIPlayerShortVideoUIManager

您可以通过 TUIPlayerShortVideoUIManager 的接口 和 面向协议控制层界面去实现您的自定义 UI。参见如下代码：

```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc]
init];
[uiManager setControlViewClass: TUIPSControlView.class];
[uiManager setLoadingView:[[TUIPSLoadingView alloc] init]];
[uiManager setBackgroundView:[UIView new]];
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

上面的代码通过 TUIPlayerShortVideoUIManager 自定义了名为 TUIPSControlView 的视频控制层（进度条，时间等），以及名为 TUIPSLoadingView 的 loading 加载控件。

TUIPlayerShortVideoUIManager 的接口参见下表：

参数名称	含义
setLoadingView	设置加载图。
setBackgroundView	设置背景图。
setErrorView	设置错误界面。
setControlViewClass	设置视频控制层* @param ViewClass 控制层类，ViewClass 是您封装好的视频控制 View，包含如进度条，时间 lable 等控件 * 它将被整体覆盖在视频窗口上，大小与视频窗口一致。
setControlViewClass :viewType	设置不同类型的视频控制层。
getLoadingView	获取加载图实 View 实例。
getBackgroundView	获取背景图 View 实例。
getErrorView	获取错误界面 View 实例。
getControlViewClass	获取视频控制界面 View 类。
getControlViewClassWith ViewType	获取不同类型视频控制界面类。

播放层目前支持的类型有两种：

- TUI_ITEM_VIEW_TYPE_VOD ///视频
- TUI_ITEM_VIEW_TYPE_LIVE ///直播
- TUI_ITEM_VIEW_TYPE_CUSTOM /// 自定义类型（例如广告页面）

相应的控制层的协议也支持两种：

- TUIPlayerShortVideoControl ///视频控制层协议
- TUIPlayerShortVideoLiveControl ///直播控制层协议
- TUIPlayerShortVideoCustomControl ///自定义控制层协议

> **说明：**
> 所有非点播和直播的界面，均以 TUI_ITEM_VIEW_TYPE_VOD 和 TUIPlayerShortVideoCustomControl 去呈现，Custom 是一个大类，具体的细分需要用户在此白板上自行定义，如上文 TUIPlayerDataModel 模型构建提到的，可以通过 extInfo/type 或者 extInfo 下别的字段去做更细粒度的划分。

TUIPlayerShortVideoControl 协议具体接口说明如下：

参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互。
model	当前播放的视频模型。

currentPlayerStatus	当前播放器的播放状态。
showCenterView	显示中心 view。
hideCenterView	隐藏中心 view。
showLoadingView	显示 loading 图。
hiddenLoadingView	隐藏 loading 图。
setDurationTime	总的视频时长。
setCurrentTime	当前的播放时长。
setProgress	进度条进度。
showSlider	显示进度条。
hideSlider	隐藏进度条。
reloadControlData	触发视图刷新。
getPlayer	获取播放器对象。
onPlayEvent	获取播放器事件。
getVideoLayerRect	获取视频渲染区域的变化。
getVideoWidget	获取视频渲染图层对象。
onPlayer:subtitleData:	播放器回调字幕，外部可自行渲染字幕信息。

TUIPlayerShortVideoLiveControl 协议具体接口说明如下：

参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互。
model	当前播放数据模型。
reloadControlData	触发视图刷新。
getPlayer	获取播放器对象。
getVideoLayerRect	获取视频图层区域。
getVideoWidget	获取视频渲染图层。

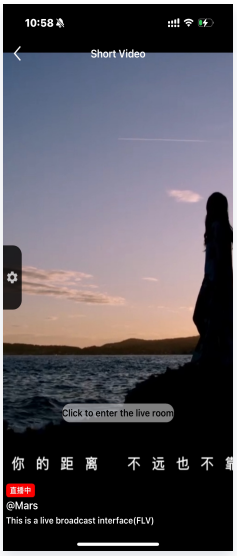
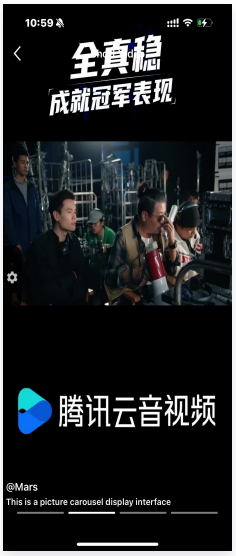
TUIPlayerShortVideoCustomControl 协议具体接口说明如下：

参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互。
model	当前播放数据模型。

reloadControlData	触发视图刷新。

> **注意：**
> 传入的自定义控制层 View 需要遵守相关的协议，否则将会编译失败。

5.3. 示例

类型	点播	直播	Custom（轮播图）	Custom（图文广告）
样例				

5.3.1. 定义不同的样式下的 UI 样式

点播样式（TUIPlayerShortVideoControl）

```
@interface TUIPSControlView : UIView<TUIPlayerShortVideoControl>
@property (nonatomic, strong) TUIPlayerVideoModel *videoModel;
@end
@implementation TUIPSControlView

-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
}

-(void)setModel:(TUIPlayerVideoModel *)model {
    _model = model;
    /// 数据
```

```
}
@end
```

直播样式(TUIPlayerShortVideoLiveControl)

```
@interface TUIPSControlLiveView : UIView<TUIPlayerShortVideoLiveControl>
@property (nonatomic, strong) TUIPlayerLiveModel *videoModel;
@end
@implementation TUIPSControlLiveView
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
}

-(void)setModel:(TUIPlayerLiveModel *)model {
    _model = model;
    /// 数据
}
@end
```

轮播图&图文广告等其他样式(TUIPlayerShortVideoCustomControl)

```
@interface TUIPSControlCustomView : UIView<TUIPlayerShortVideoCustomControl>
@property (nonatomic, strong) TUIPlayerDataModel *videoModel;
@end
@implementation TUIPSControlCustomView
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
}

-(void)setModel:(TUIPlayerDataModel *)model {
    _model = model;
    /// 数据

    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];
```

```
    if ([type isEqualToString:@"ad"]) {  ///图文广告
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:adUrl]]];
    } else if ([type isEqualToString:@"imageCycle"]) { ///轮播图
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }

}
@end
```

5.3.2. 将创建好的样式通过 TUIPlayerShortVideoUIManager 注册

```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc]
init];
[uiManager setControlViewClass: TUIPSControlView.class
viewType:TUI_ITEM_VIEW_TYPE_VOD];
[uiManager setControlViewClass: TUIPSControlLiveView.class
viewType:TUI_ITEM_VIEW_TYPE_LIVE];
[uiManager setControlViewClass: TUIPSControlCustomView.class
viewType:TUI_ITEM_VIEW_TYPE_CUSTOM];
```

5.3.2. 通过 TUIPlayerShortVideoUIManager 初始化 TUIShortVideoView

```
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

5.3.3. 通过 setShortVideoStrategyModel 和 setShortVideoLiveStrategyModel 设置点播和直播的相关策略

```
// Set your playback strategy
 TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
 model.mPreloadConcurrentCount = 1;
 model.preDownloadSize = 1;
 model.enableAutoBitrate = NO;
[_videoView setShortVideoStrategyModel:model];

 // live strategy
 TUIPlayerLiveStrategyModel *liveStrategyModel = [[TUIPlayerLiveStrategyModel alloc]
init];
```

```
 liveStrategyModel.isLastPrePlay = YES;
 [_videoView setShortVideoLiveStrategyModel:liveStrategyModel];
```

5.3.4. UI 与数据的关系

TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 可以理解为预置模板，初始化 TUIShortVideoView 的时候已经预置好了各种类型的模板，当滑倒相对应的数据类型的时候，即会显示当前模板。

类型	数据模型	UI 模板
点播	TUIPlayerVideoModel	TUIPSControlView
直播	TUIPlayerLiveModel	TUIPSControlLiveView
自定义类型（轮播图，图文广告等）	TUIPlayerDataModel	TUIPSControlCustomView

- UI 模板是提前预置的。
- UI 样式的显示则根据相对应的数据类型的驱动。

5.3.4. UI 模板与 TUIShortVideoView 的交互

自定义 UI 模板通过对应的协议与 TUIShortVideoView 实现交互 TUIShortVideoView 向 TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 传递消息通过 TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl 的协议方法。
点播：

```
-(void)setModel:(TUIPlayerVideoModel *)model {

    if ([_model observationInfo]) {

        [_model removeObserver:self forKeyPath:@"preloadState"];

    }
    _model = model;
    [model addObserver:self forKeyPath:@"preloadState"
options:NSKeyValueObservingOptionNew context:nil];


    NSDictionary *dic = model.extInfo;
    NSString *iconUrl = [dic objectForKey:@"iconUrl"];
    NSString *advertise = [dic objectForKey:@"advertise"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *title = [dic objectForKey:@"title"];
    NSString *topic = [dic objectForKey:@"topic"];
    self.iconImageView.image = [UIImage imageNamed:iconUrl];
    [self.adButton setTitle:advertise forState:UIControlStateNormal];
    self.nameLabel.text= name;
    self.themeLabel.text = topic;
    self.desLabel.text = title;
```

```
    [self updatePreloadState];
    [self updateLickCount];
    model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
        [self updateLickCount];
    };
}
```

直播：

```
-(void)setModel:(TUIPlayerLiveModel *)model {

    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"liveTitile"];
    NSString *adDes = [dic objectForKey:@"liveDes"];
    NSString *name = [dic objectForKey:@"name"];

    self.nameLabel.text = name;
    self.desLabel.text = adTitile;
}
```

Custom（轮播图，图文等）：

```
-(void)setModel:(TUIPlayerDataModel *)model {

    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];
    self.desLabel.text = adTitile;
    self.nameLabel.text = name;
    if ([type isEqualToString:@"web"]) {
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:adUrl]]];
    } else if ([type isEqualToString:@"imageCycle"]) {
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
```

```
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }
}
```

TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 向 TUIShortVideoView 传递消息则通过 TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl 的 delegate。

点播：

```
@protocol TUIPlayerShortVideoControlDelegate <NSObject>

/**
 * 暂停
 */
- (void)pause;
/**
 * 继续播放
 */
- (void)resume;

/**
 * 滑动滚动条的处理
 * @param time    滑动的距离
 */
- (void)seekToTime:(float)time;

/**
 * 是否正在播放
 */
- (BOOL)isPlaying;

/**
 * 重置视频播放容器
 * - 用于视频播放容器被移除后需要重置的场景
 */
- (void)resetVideoWeigetContainer;

@optional
/**
 * 自定义回调事件
 */
- (void)customCallbackEvent:(id)info;
@end
```

```
/////调用
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

直播：

```
@protocol TUIPlayerShortVideoLiveControlDelegate <NSObject>

/**
 * 暂停
 */
- (void)pause;

/**
 * 继续播放
 */
- (void)resume;

/**
 * 重置视频播放容器
 * - 用于视频播放容器被移除后需要重置的场景
 */
- (void)resetVideoWeigetContainer;

@optional
/**
 * 自定义回调事件
 */
- (void)customCallbackEvent:(id)info;
@end

/////调用
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

Custom（轮播图，图文等）：

```
@protocol TUIPlayerShortVideoCustomControlDelegate <NSObject>

@optional
/**
 * 自定义回调事件
```

```
 */
- (void)customCallbackEvent:(id)info;
@end

/////调用
if (self.delegate && [self.delegate
respondsToSelector:@selector(customCallbackEvent:)]) {
    [self.delegate customCallbackEvent:@"test"];
}
```

说明：
完整示例请看 Demo。

高级功能

业务通知消息到页面图层

TUI 提供了消息接口供用户把数据实时通知到当前图层，可以通过数据操作对象获取到视频对象之后，进行通知，示例如下：

```
/// 获取数据管理器
TUIShortVideoDataManager *dataManager = [self.videoView getDataManager];
///获取数据模型
TUIPlayerDataModel *model = [dataManager getDataByPageIndex:1];
///修改数据模型
model.extInfo = @{@"key":@"value"}
///通知数据模型发生改变
[model extInfoChangeNotify];
```

随后在 UI 控制层的 onExtInfoChanged 回调中会收到该通知，从而对当前页面进行 UI 修改，示例如下：

```
model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
      [self updateLickCount];
};
```

注意：
此功能只对 extInfo 字段有效

点播设置音量均衡

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

注意：

播放器高级版 11.7 版本开始支持。

```
/// 音量均衡 .响度范围：-70～0(LUFS)。此配置需要LiteAVSDK 11.7 及以上版本支持。
/// 以下几种常量供参考使用
/// 关：AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h)
/// 开（标准响度）：AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h)
/// 开（低响度）：AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h)
/// 开（高响度）：AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h)
/// 默认值为AUDIO_NORMALIZATION_OFF。

TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
model.audioNormalization = AUDIO_NORMALIZATION_STANDARD;

[_videoView setShortVideoStrategyModel:model];
```

点播终端极速高清

TUI 短视频支持开启终端高清，终端极速高清介绍可参见 终端极速高清 。
根据文档中的步骤，将超分插件下载下来，集成到您的项目中。随后在设置点播策略的时候，开启超分策略。

```
TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
model.superResolutionType = TUI_SuperResolution_ASR;

[_videoView setShortVideoStrategyModel:model];
```

画中画组件（TUIPIP）
iOS

最近更新时间：2024-11-05 15:22:12

应用场景

高级画中画是在原 基础画中画 上进行的升级，主要支持加密视频画中画、离线播放画中画、从前台无缝切换到画中画的场景，优化了实现方式和逻辑，无需长时间等待，实现真正意义的“秒切”效果。
高级画中画优势：

- 加密视频画中画：和现有播放器加密播放完美结合，实现基于加密模板的视频画中画播放，无需切换播放器类型。
- 离线播放画中画：支持本地视频画中画播放，包含普通视频、加密视频等。
- “秒切”效果：无需点击切换画中画按钮，退后台即可立马启动画中画，实现真正意义的“秒切”。

环境要求

系统版本：iOS >= 14.0、iPad >= 9.0
硬件设备：iPhone 8及以上的设备
SDK 版本：11.4版本及以上

集成步骤

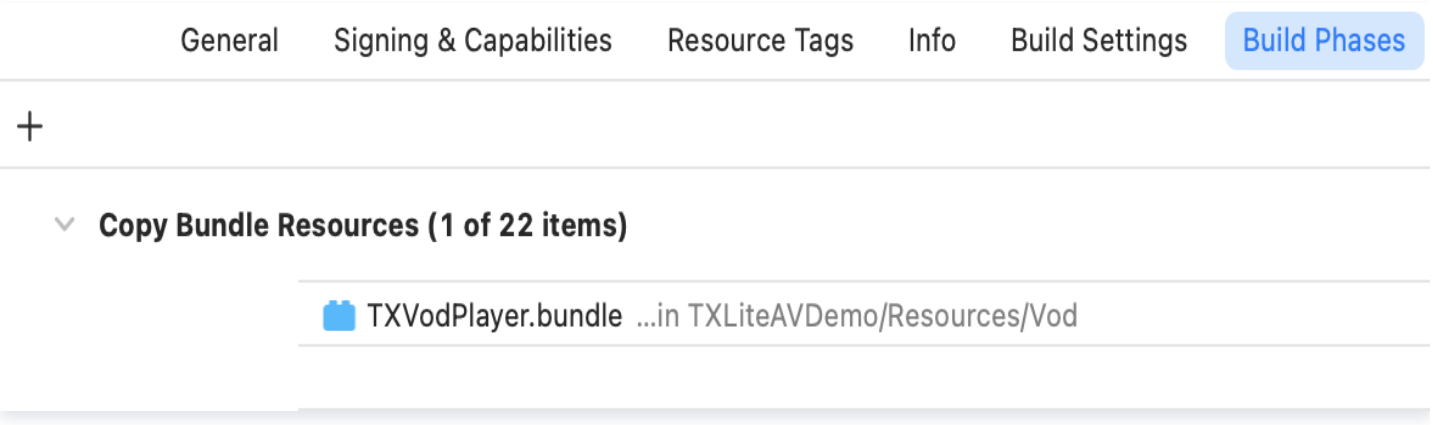
升级版本及配置资源

1. 升级 SDK 版本

高级画中画需要 SDK 配合使用，在使用高级画中画版本功能前需要将 **SDK 的版本升级到11.3及以上的高级版本 或 11.4及以上的基础版本**，否则无法使用。同时，基础画中画 版本和高级画中画版本两者可以兼容性的存在，不会存在功能性冲突。若想升级 SDK 版本，请参见 SDK 集成指引 。

2. 引入 bundle 资源

因为 SDK 内需要使用 TXVodPlayer.bundle 里的资源，需在编译之前将 bundle 文件 下载 引入到项目中，切勿更改 bundle 及其内部使用的资源名称，否则会导致无缝切换画中画失败。



3. 开通播放器高级版 Licence

高级画中画版本需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引获取。若您已获取对应 License，可前往 腾讯云视立方控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请 Player 高级套餐 License，进入画中画将无效。
获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中进行如下设置：

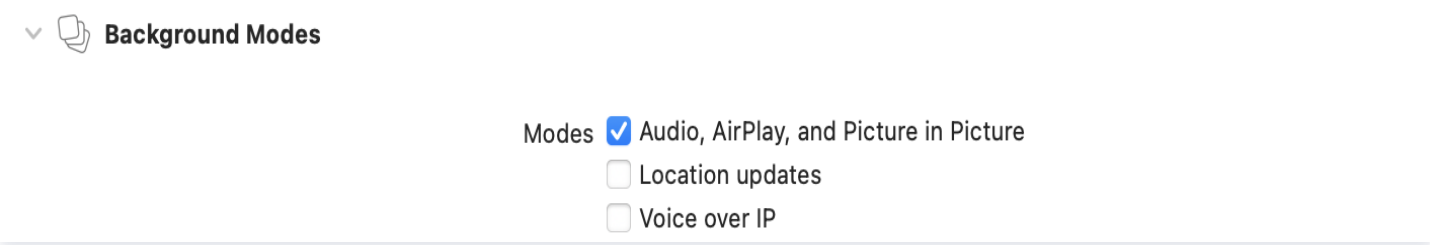
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<获取到的 licenseUrl>";
    NSString * const licenceKey = @"<获取到的 key>";

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

画中画功能快速接入

1. 权限开通

画中画（PictureInPicture）在 iOS 9 就已经推出了，不过之前都只能在 iPad 上使用，iPhone 要使用画中画需更新到 iOS 14 才能使用。 目前腾讯云播放器可以支持应用内和应用外画中画能力，极大的满足用户的诉求。使用前需要开通后台模式，步骤为：**XCode 选择对应的 Target > Signing & Capabilities > Background Modes，勾选 “Audio, AirPlay, and Picture in Picture”** 。



2. 设置配置选项

为了使用自动画中画功能，需要在设置中打开**自动开启画中画**按钮。具体路径为 iPhone 或 iPad上选择：**设置 > 通用 > 画中画 > 自动开启画中画**，选择打开即可。

3. 设置代理

为了便于监听画中画的状态，需要设置 vodDelegate，实现 TXVodPlayListener 中的画中画相关回调。可以根据回调里的各种状态和错误信息，进行相关的业务操作，例如：继续播放、暂停或退出画中画等。

```
/**
 * 画中画状态回调
 *
 * 画中画状态回调
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureStateDidChange:
(TX_VOD_PLAYER_PIP_STATE)pipState withParam:(NSDictionary *)param;

/**
 * 画中画错误信息回调
 *
 * 画中画错误信息回调
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureErrorDidOccur:
(TX_VOD_PLAYER_PIP_ERROR_TYPE)errorType withParam:(NSDictionary *)param;
```

4. 使用画中画能力代码示例

注意：

- 使用自动画中画功能一定要确保播放器处于播放状态，若播放器是暂停或停止状态时，无法使用自动画中画功能。
- isSupportSeamlessPictureInPicture 这个接口，需要在应用程序加载高级版 License 以后才能使用。同时，此接口只能判断设备本身是否支持自动切换画中画，因系统限制，无法判断用户对于自动画中画的设置权限，需自行引导。
- 播放之前先设置是否允许“自动切换画中画功能”

```
// 1.播放之前先设置“自动切换 Picture-In-Picture功能”是否允许
// YES 表示允许 NO 表示不允许，默认为NO
[TXVodPlayer setPictureInPictureSeamlessEnabled:YES];
```

```
// 2、进入画中画
 if (![TXVodPlayer isSupportPictureInPicture]) {
   // 设备不支持画中画直接退出
    return;
 }

// 手动调用进入画中画
 [_vodPlayer enterPictureInPicture];

// 3、退后台操作 如果设备支持无缝切换画中画，退后台不暂停播放。
// 注意：isSupportSeamlessPictureInPicture这个接口，需要在应用程序加载高级版License以后才能
使用。同时，此接口只能判断设备本身
// 是否支持自动切换画中画，因系统限制，无法判断用户对于自动画中画的设置权限，需自行引导。
if ([self.vodplayer isSupportSeamlessPictureInPicture]) {
     // 不做处理
} else {
   // 暂停播放
   [self.vodplayer pause];
}

// 4.退出画中画
[_vodPlayer exitPictureInPicture];
```

VR 播放插件

最近更新时间：2025-03-04 17:03:13

功能简介

VR 播放组件可用于 VR 全景视频播放，播放中可以通过陀螺仪转动或手势操作来改变视角，360度无死角的观看全景视频。目前 VR 播放组件可支持配置单目或双目模式，单目模式适用于裸眼观看全景视频，双目模式适用于 VR 眼镜等设备观看。此外 iOS 端支持180度半球模型全景视频，可适应更多的使用场景。

使用条件

- 目前移动端播放器 SDK 11.3 以上版本及移动端播放器 SDK（高级版）均可使用 VR 播放插件。
- VR 播放需获取 移动端播放器高级版 License 方可使用。

操作步骤

步骤1：获取插件

1. VR 播放插件（下文简称“插件”）作为独立的 SDK 提供，您可以根据项目需求进行集成，下载地址如下：

终端类别	VR 插件下载地址
Android 端	VR 插件
iOS 端	VR 插件

2. 将下载的插件 plugin_monet-release-v.x.x.x.aar 或 TXCMonetPlugin-release-v.x.x.x.framework（x.x.x. 为版本号）集成到工程，启动播放器实例后，宿主会自动加载插件。

步骤2：初始化 License

使用 VR 播放功能需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引获取。若您已获取对应 License，可前往 腾讯云视立方控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey，随后通过下面的接口进行初始化 License。

Android

```
String licenceUrl = "填入您购买的 License 的 URL";
String licenseKey = "填入您购买的 License 的 Key"
TXLiveBase.getInstance().setLicence(context, licenceUrl, licenseKey);
```

iOS

```
NSString *licenceURL = "填入您购买的 License 的 URL";
NSString *licenseKey = "填入您购买的 License 的 Key"
[TXLiveBase setLicenceURL:licenceURL key:licenseKey];
```

步骤3：开启或关闭终端 VR 全景视频能力

启动播放后（建议在收到 TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED 事件后调用），可以通过 TXVodPlayer 下面的接口，进行开启和关闭 VR 全景视频：

Android

```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
        //开启终端VR全景视频单目模式，如需开启双目模式，value设为“12”即可
        mVodPlayer.setStringOption("PARAM_MODULE_TYPE", 11);
    }


}


//关闭终端VR全景视频
mVodPlayer.setStringOption("PARAM_MODULE_TYPE", 0);
```

iOS

```
//开启VR全景视频单目模式，如需开启双目模式value设为“12”
//注意：VR全景视频的开启设置需要在播放器播放之前，或者如果在播放过程中进行设置，在设置以后需要重启一
下
//播放，建议这里可以做一个切换动画
NSMutableDictionary *extInfoMap = [NSMutableDictionary dictionary];
[extInfoMap setObject:@"11" forKey:@"PARAM_MODULE_TYPE"];
[_txVodPlayer setExtentOptionInfo:extInfoMap];

//关闭VR全景视频：
[extInfoMap setObject:@"0" forKey:@"PARAM_MODULE_TYPE"];
```

插件加载配置

终端极速高清插件加入到工程后，默认会自动加载， 如果您不希望播放器加载该插件，可以通过下面的方式关闭：

Android

```
TXVodPlayConfig playConfig = new TXVodPlayConfig();
playConfig.mEnableRenderProcess = false;
mVodPlayer.setConfig(playConfig);
```

iOS

```
TXVodPlayConfig *playConfig = [[TXVodPlayConfig alloc] init];
playConfig.enableRenderProcess = NO;
[_txVodPlayer setConfig:playConfig];
```

设置混淆规则

在 proguard-rules.pro 文件，将终端极速高清相关类加入不混淆名单（iOS 默认无）：

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

日志查看

Android

1. 终端 VR 全景视频插件加载成功日志：

```
D/HostEngine-PluginManger: [loadPlugin], succeed loading pluginId=2
,pluginClazzName=com.tencent.liteav.monet.MonetPlugin
```

2. 开启 VR 全景视频功能成功开启日志：

```
D/MonetPlugin-Process: [updateModule], moduleType=11
```

3. 如果出现下面的日志，表示签发的 License 不合法：

```
E/MonetPlugin-Process: [updateModule], error, reason = license is invalid!!
```

iOS

1. 终端 VR 全景视频插件加载成功日志：

```
[PluginsSDK] plugin config : pluginId = 2, pluginName = Monet
```

2. 开启终端 VR 全景视频成功开启日志：

```
[MonetProcessor] monet change VR*** module, processor: 0x10034523
```

3. 如果出现下面的日志，表示签发的 License 不合法：

```
[MonetProcessor] Monet License invalid, processor:0x10034523, set module is
```

```
null
```

终端极速高清

最近更新时间：2024-04-11 15:15:21

概述

腾讯云视立方 · 音视频终端引擎提供了**终端极速高清插件**，该插件可在终端实现视频播放的超分后处理，适用于以下场景：

成本优化

保证画质的前提下，降低视频传输成本。如在云端将 720p 视频降档为 540p 进行传输，在终端播放时利用超分技术将视频画质提升为 720p 效果进行播放。

画质提升

将原本低分辨率的视频转化为高分辨率视频，提升视频播放清晰度，实现画质优化。

> **注意：**
> - 目前仅腾讯云视立方 · 播放器 SDK（Player）移动端9.5.29009以上版本可使用该插件。
> - 使用终端极速高清插件需要申请 License，当前提供免费的测试版 License，获取方式请参见 终端极速高清 License 。

效果和功耗

可单击下方视频查看终端极速高清功能效果，视频左半部分为原视频，右半部分为开启终端极速高清功能后的效果：

观看视频

开启终端极速高清后，近景处路面碎石、远方树叶等清晰度均得到提升。

> **说明：**
> 运行终端极速高清插件后，电流提升小于100mA，额外耗电几乎可以忽略。

操作步骤

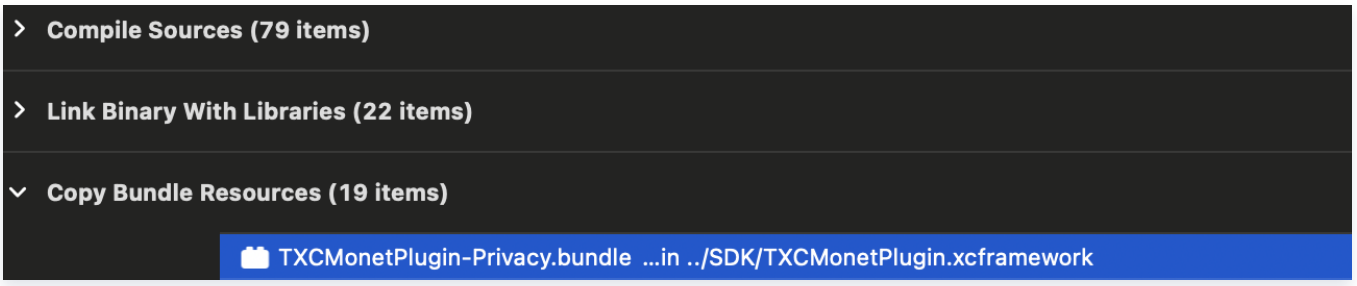
步骤1：获取插件

1. 终端极速高清插件（下文简称“插件”）作为独立的 SDK 提供，您可以根据项目需求进行集成，下载地址如下：

终端类别	SDK 下载地址
Android 端	SDK
iOS 端	SDK

2. 将下载的插件 plugin_monet-release-v.x.x.x.aar 或 TXCMonetPlugin-release-v.x.x.x.framework（x.x.x. 为版本号）集成到工程，启动播放器实例后，宿主会自动加载插件。
3. 对于 iOS 端，在集成超分 SDK 后，需要在项目工程中添加 CoreML.framework 库，路径为"Build Phases > Link Binary With Libraries > '选择+号添加' "即可。

同时，为了适配苹果的隐私清单，请将 TXCMonetPlugin-Privacy.bundle 添加到项目工程中，或者将 bundle 里的“PrivacyInfo.xcprivacy”整合进待集成的项目工程内。添加 TXCMonetPlugin-Privacy.bundle 到项目工程中的方法为："Build Phases > Copy Bundle Resources > '选择+号添加'"即可。



步骤2：加载资源

Android 端超分需要额外集成超分模型资源，资源 下载地址 。下载可以通过集成到 app assets 目录或者动态加载方法进行配置资源。必须确保资源配置成功，才可以进行初始化 License。

集成到 assets 目录

1. 解压下载后的 sr_resource.zip 文件， 把 sr_resource 目录复制到 app assets 目录。
2. 在 this.getCacheDir().getAbsolutePath() 目录新建 sr_resource 文件夹，暂且称 sr_resource 的路径为 dstPath， 然后把解压后的 sr_resource 里的文件复制到 dstPath。
3. 配置资源加载路径： `MonetPlugin.setResourcePath(destPath)` 。

示例代码如下：

```
private void copySRResourceAsync() {
    Runnable task = new Runnable() {
        @Override
        public void run() {
            String destPath = MainActivity.this.getCacheDir().getAbsolutePath()
+ "/sr_resource";
            File file = new File(destPath);
            if (!file.exists()) {
                file.mkdir();
            }
            copyAssetsToDst(MainActivity.this, "sr_resource", destPath);
            MonetPlugin.setResourcePath(destPath);
        }
    };
```

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

private static void copyAssetsToDst(Context context, String srcDir, String
dstDir) {
    String fileNames[] = null;
    try {
        fileNames = context.getAssets().list(srcDir);
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    if (fileNames.length > 0) {
        File file = new File(dstDir);
        if (!file.exists()) {
            file.mkdirs();
        }
        for (String fileName : fileNames) {
            if (!srcDir.equals("")) {
                copyAssetsToDst(context, srcDir + File.separator + fileName,
                        dstDir + File.separator + fileName);
            } else {
                copyAssetsToDst(context, fileName, dstDir + File.separator +
fileName);
            }
        }
    } else {
        InputStream is = null;
        FileOutputStream fos = null;
        try {
            File outFile = new File(dstDir);
            is = context.getAssets().open(srcDir);
            fos = new FileOutputStream(outFile);
            byte[] buffer = new byte[8192];
            int byteCount;
            while ((byteCount = is.read(buffer)) != -1) {
                fos.write(buffer, 0, byteCount);
            }
            fos.flush();
            is.close();
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (fos != null) {
                    fos.flush();
                }
```

```
                if (is != null) {
                    is.close();
                }
                if (fos != null) {
                    fos.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

动态加载

1. 把下载的 sr_resource.zip 文件上传到贵司私有的服务器， 不建议直接使用腾讯云提供的下载地址，此地址指向文件会不定期更新。App 启
   动后， 把 sr_resource.zip 下载到 Sdcard 并解压为 sr_resource。
2. 在 this.getCacheDir().getAbsolutePath() 目录新建 sr_resource 文件夹，暂且称 sr_resource 的路径为 dstPath， 然后把 Sdcard 上解压后的 sr_resource 里的文件复制到 dstPath。
3. 配置资源加载路径： `MonetPlugin.setResourcePath(destPath)` 。相关示例代码可以参见 集成到 assets 章节的指引 。

iOS 端超分无需此步骤，直接集成 framework 即可。

步骤3：初始化 License

1. 使用终端极速高清插件需要申请 License，当前提供免费的测试版 License，获取方式请参见 终端极速高清 License 。
2. 获取到 License 后，通过下面的接口进行初始化 License：

Android

```
String licenceUrl = "填入您购买的 License 的 URL";
String licenseKey = "填入您购买的 License 的 Key"
TXLiveBase.getInstance().setLicence(context, licenceUrl, licenseKey);
```

iOS

```
NSString *licenceURL = "填入您购买的 License 的 URL";
NSString *licenseKey = "填入您购买的 License 的 Key"
[TXLiveBase setLicenceURL:licenceURL key:licenseKey];
```

步骤4：开启或关闭终端极速高清能力

启动播放之后（建议在收到 TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED 事件之后调用），可以通过 TXVodPlayer 下面的接口，进行开启和关闭超分：

Android

```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
        //开启终端极速高清
        mVodPlayer.setStringOption("PARAM_SUPER_RESOLUTION_TYPE", 1);
    }


}


//关闭终端极速高清
mVodPlayer.setStringOption("PARAM_SUPER_RESOLUTION_TYPE", 0);
```

iOS

```
//开启超分
//注意：超分的开启设置需要在播放器播放之前，或者如果在播放过程中进行设置，在设置以后需要重启一下
//播放，建议这里可以做一个切换动画
NSMutableDictionary *extInfoMap = [NSMutableDictionary dictionary];
[extInfoMap setObject:@"1" forKey:@"PARAM_SUPER_RESOLUTION_TYPE"];
[_txVodPlayer setExtentOptionInfo:extInfoMap];

//关闭超分：
[extInfoMap setObject:@"0" forKey:@"PARAM_SUPER_RESOLUTION_TYPE"];
```

插件加载配置

终端极速高清插件加入到工程后，默认会自动加载， 如果您不希望播放器加载该插件，可以通过下面的方式关闭：

Android

```
TXVodPlayConfig playConfig = new TXVodPlayConfig();
playConfig.mEnableRenderProcess = false;
mVodPlayer.setConfig(playConfig);
```

iOS

```
TXVodPlayConfig *playConfig = [[TXVodPlayConfig alloc] init];
playConfig.enableRenderProcess = NO;
```

```
[_txVodPlayer setConfig:playConfig];
```

设置混淆规则

在 proguard-rules.pro 文件，将终端极速高清相关类加入不混淆名单（iOS 默认无）：

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

日志查看

Android

1. 终端极速高清插件加载成功日志：

```
D/HostEngine-PluginManger: [loadPlugin], succeed loading pluginId=2
,pluginClazzName=com.tencent.liteav.monet.MonetPlugin
```

2. 开启终端极速高清功能成功开启日志：

```
D/MonetPlugin-Process: [updateModule], moduleType=1
```

3. 如果出现下面的日志，表示签发的 License 不合法：

```
E/MonetPlugin-Process: [updateModule], error, reason = license is invalid!!
```

iOS

1. 终端极速高清插件加载成功日志：

```
[PluginsSDK] plugin config : pluginId = 2, pluginName = Monet
```

2. 开启终端极速高清功能成功开启日志：

```
[MonetProcessor] PLUGIN: did update monet module, result = 1
```

3. 如果出现下面的日志，表示签发的 License 不合法：

```
[MonetProcessor] Monet License invalid, error, set module is null
```

播放质量监控
点播播放数据

最近更新时间：2024-11-25 15:02:02

点播播放质量监控提供点播播放全链路的数据统计、质量监控及可视化分析服务。支持实时数据上报、数据聚合、多维筛选和精细化定向分析，可帮助企业实时掌控大盘运营状况、了解用户习惯和行为特征，有效指导运营决策、驱动业务增长。

注意事项

- 点播播放质量监控仅适用于使用 **腾讯云视立方 SDK** 的场景，其他 SDK 无法进行监控。
- 点播播放质量监控仅适用于**腾讯云点播的视频资源**，即使用腾讯云点播文件标识 FileID 进行播放的视频资源，其他视频来源无法识别对应腾讯云用户身份。

> **注意：**
> 在音视频终端 SDK（腾讯云视立方）中，Web 端仅使用腾讯云视立方 · 播放器 SDK TCPlayer 4.4.0 及其以后的版本可获取播放数据；移动端仅使用腾讯云视立方 · 播放器 SDK 9.5.29006 及其以后的版本可获取播放数据，短视频 SDK、全功能 SDK 等其他 SDK 后续也将在9.6版本陆续支持，敬请期待。

版本说明

点播播放质量监控服务分为基础版和高级版，不同版本差异见下表：

版本	数据存储时间	费用
基础版	3天	免费
高级版	1年	体验期暂不收费，详情请参见 点播播放质量监控计费说明 。

操作步骤

步骤1：开通点播播放质量监控服务

1. 登录 腾讯云视立方控制台，在左侧菜单栏选择 **SDK 质量监控**下的 **点播播放数据**。

2. 阅读并确认服务开通界面说明，勾选“我已知晓并同意《SDK隐私协议》和《点播播放质量监控计费说明》”，单击**立即开通**。

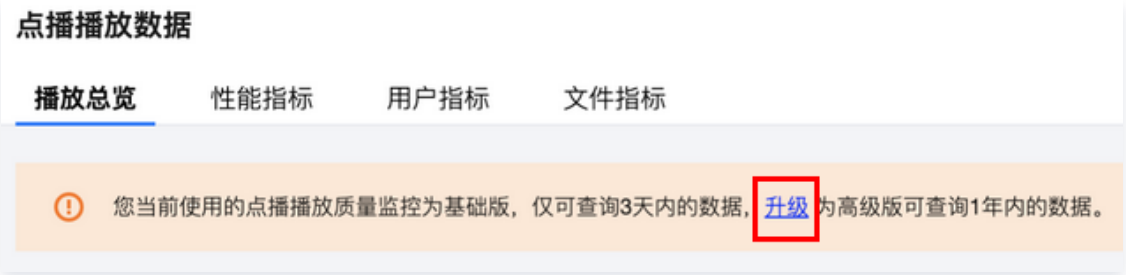
3. 成功开通的点播播放质量监控服务默认为**基础版**，仅可查询3天内的数据。

> **说明：**
> 若需升级为高级版，可参见 将基础版升级为高级版 。
>
>
>

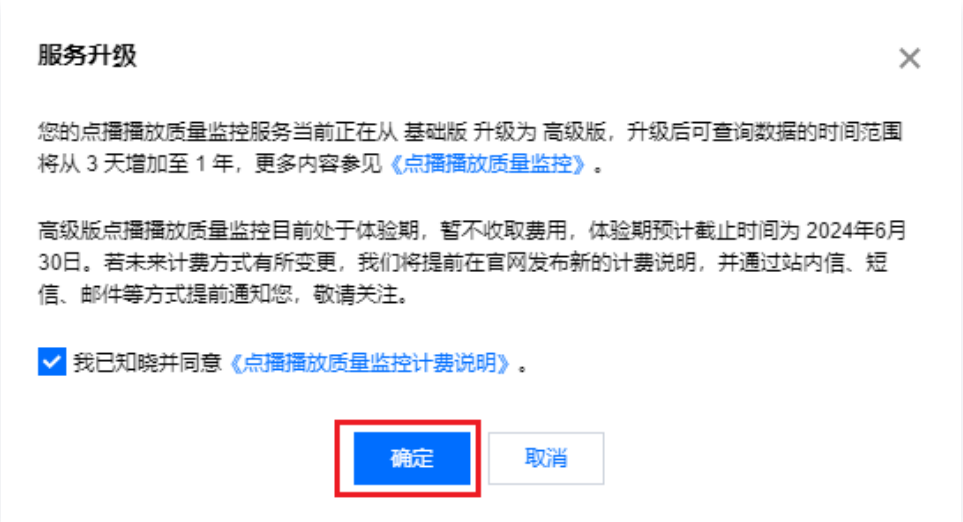
步骤2：将基础版升级为高级版

> **注意：**
> 高级版点播播放质量监控目前处于体验期，暂不收取费用，体验期预计截止时间为2024年06月30日。若未来计费方式有所变更，我们将提前在官网发布新的计费说明，并通过站内信、短信、邮件等方式提前通知您，敬请关注。

1. 进入**点播播放数据**，单击上方的**升级**，打开服务升级弹框。



2. 阅读并确认服务升级说明后，勾选“我已知晓并同意《点播播放质量监控计费说明》”，单击**确定**即可。



3. 升级为高级版后，右侧的版本将显示为“高级版”。

> **说明：**
> 不同标签页数据指标详见 数据指标，数据查看功能使用说明详见 功能说明。

数据指标

播放总览

播放总览可了解当前播放总量和用户规模。

指标	说明
播放次数	播放的总次数。 视频首次发起播放请求记为一次播放，中途暂停后继续播放仍然算一次播放。
独立用户数	用户总数。 移动端根据设备 ID 识别独立用户，Web 端根据 UUID 识别独立用户。

性能指标

性能指标可了解当前 SDK 的服务质量。

指标	说明
平均首帧时长	平均首帧时长 = 所有首帧时长总和 / 播放次数 首帧时长：用户从发起播放请求到首帧完成播放的时间。
平均卡顿率	平均卡顿率 = 所有播放卡顿率总和 / 播放次数 播放卡顿率：单次播放过程中出现卡顿的总时长占播放时长的比例。 卡顿：播放过程中出现视频暂停加载（loading）的情况，不包含拖动进度条造成的视频加载。
播放失败率	播放失败次数占总播放次数的比例。 播放失败次数为有错误码返回的播放次数。

用户指标

用户指标可了解用户行为特征和观看习惯。

指标	说明
人均观看时长	人均观看视频的时长，即总播放时长/总独立用户数。
人均播放次数	人均发起播放请求的次数，即总播放次数/总独立用户。
播放次数 TOP 100 视频	播放次数前 100 的文件，包含排名、ID、播放次数、人均观看时长和人均播放次数。

文件指标

文件指标可查看单个文件的播放情况，使用云点播文件标识 FileID 进行查看。

指标	说明
文件基本信息	视频 ID、视频名称、视频时长、平均播放时长、播放总次数。
播放次数	当前文件的播放总次数。
平均播放时长	当前文件平均播放时长，即文件的总播放时长/文件的播放次数。
播放次数排名	当前文件在所有播放过的文件中，播放次数的排名。

版权所有：腾讯云计算（北京）有限责任公司

功能说明

点播播放质量监控服务包含基本功能、数据对比和高级筛选三种数据查看能力。

基本功能

功能	说明
平台	包含 iOS、Android 和 Web 三种平台选项，为必选项。
应用	以包名作为应用唯一标识。选项与平台相关： • 当平台为 iOS 或 Android 时，应用为必选项。 • 当平台选择为 Web 时，默认无应用，此时不可选。
查询时间	选择查看时间段，可使用快捷时间也可自定义数据查询时间段，为必选项。

注意：
播放总览、性能指标和用户指标的查询时间最小粒度为5分钟，文件指标的查询时间最小粒度为1天。单次查询时间长度不得超过90天。

数据对比

数据对比功能可同时提供不同时间段的数据，便于进行对比分析。数据对比目前提供环比和自定义时间两种选项，二者可同时选择。

选项	说明
环比	相邻上一个时间周期。
自定义时间	可自定义需要进行对比查看的时间。

注意：
数据对比选项的时间长度与查询时间一致。因此在选择自定义时间的起始时间后，将自动根据查询时间的时间长度计算终止时间。
示例：选择的查询时间为2021-07-08 12:00:00 至 2021-07-14 14:00:00，则时间长度为6天2小时。
此时环比时间为2021-07-02 9:55:00 至 2021-07-08 11:55:00。若选择的自定义起始时间为2021-06-01 01:00:00，则默认终止时间为2021-06-07 03:00:00。

高级筛选

高级筛选提供多种数据维度，便于分析特殊因素对于数据的影响。具体如下：

选项	说明
域名	播放视频所使用的域名。
地域	视频播放请求发起者所在地域。
运营商	播放视频所使用的运营商。
操作系统	播放视频使用的操作系统。

版权所有：腾讯云计算（北京）有限责任公司

网络环境	播放视频所使用的网络环境，仅当平台为 iOS 或 Android 时可选。
浏览器	播放视频所使用的浏览器，仅当平台为 Web 时可选。
SDK版本	播放视频所使用的腾讯云视立方 SDK 版本。

SDK 质量监控

最近更新时间：2024-10-28 18:02:21

注意事项

- 统计分析仅适用于使用腾讯云的直播 SDK，其它 SDK 无法监控。
- 推流数据查询只能查询最近3天，且时间跨度在6小时之内。

操作步骤

1. 登录 云直播控制台，在左侧菜单栏选择**业务监控 > SDK 质量监控**，进入质量监控页查看推流数据（**需使用直播 SDK**）。
2. 可选择查看**全局推流统计**、**全局播放统计**、**单路推流统计**、**单路播放统计**及**运营数据统计**。具体数据说明请看下文。

数据说明

全局推流统计

可查询最近3天内的设备系统数据，详细数据列表展示筛选后的 StreamName、发送码率（kbps）、发送帧率（fps）、网络运营商、推流机型、操作。

列表字段	说明
发送码率	用户查询日内该路流的平均发送码率。
发送帧率	用户查询日内该路流的平均发送帧率。
网络运营商	该路流返回数据中最近的一条，展示其网络运营商名称。
推流机型	该路流返回数据中最近的一条，展示其推流机型名称。

若您需查看单路推流，单击操作栏下的**查看**即可进行查看。

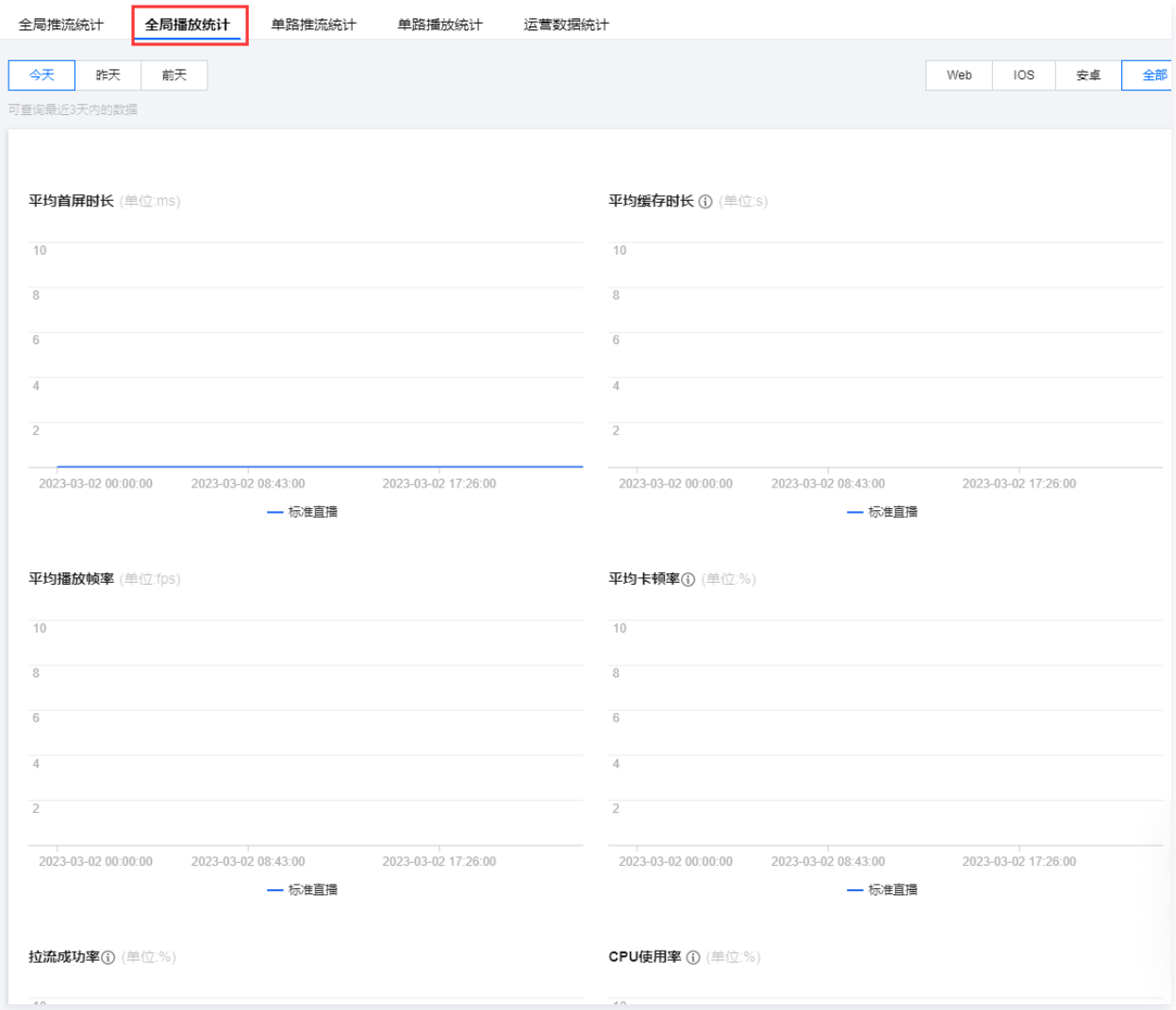
全局播放统计

可通过设定开始时间和结束时间，输入流 ID、或选择设备系统进行数据查询。数据统计包括平均首屏时长、平均缓存时长、平均播放帧率、下行网络码率、拉流成功率、在线观看人数、平均卡顿率、CPU 使用率。

统计项	说明
平均首屏时长	不同时刻单路流播放端平均首帧时长。
平均缓存时长	不同时刻单路流播放端平均缓存时长。
平均播放帧率	不同时刻单路流播放端平均播放帧率。
下行网络码率	不同时刻单路流播放端平均下行拉流码率。
拉流成功率	单路流的成功拉流数/单路流的总拉流数。
在线观看人数	不同时刻单路流播放端的拉流数。
平均卡顿率	不同时刻播放端卡顿拉流数 / 总的拉流数。

版权所有：腾讯云计算（北京）有限责任公司

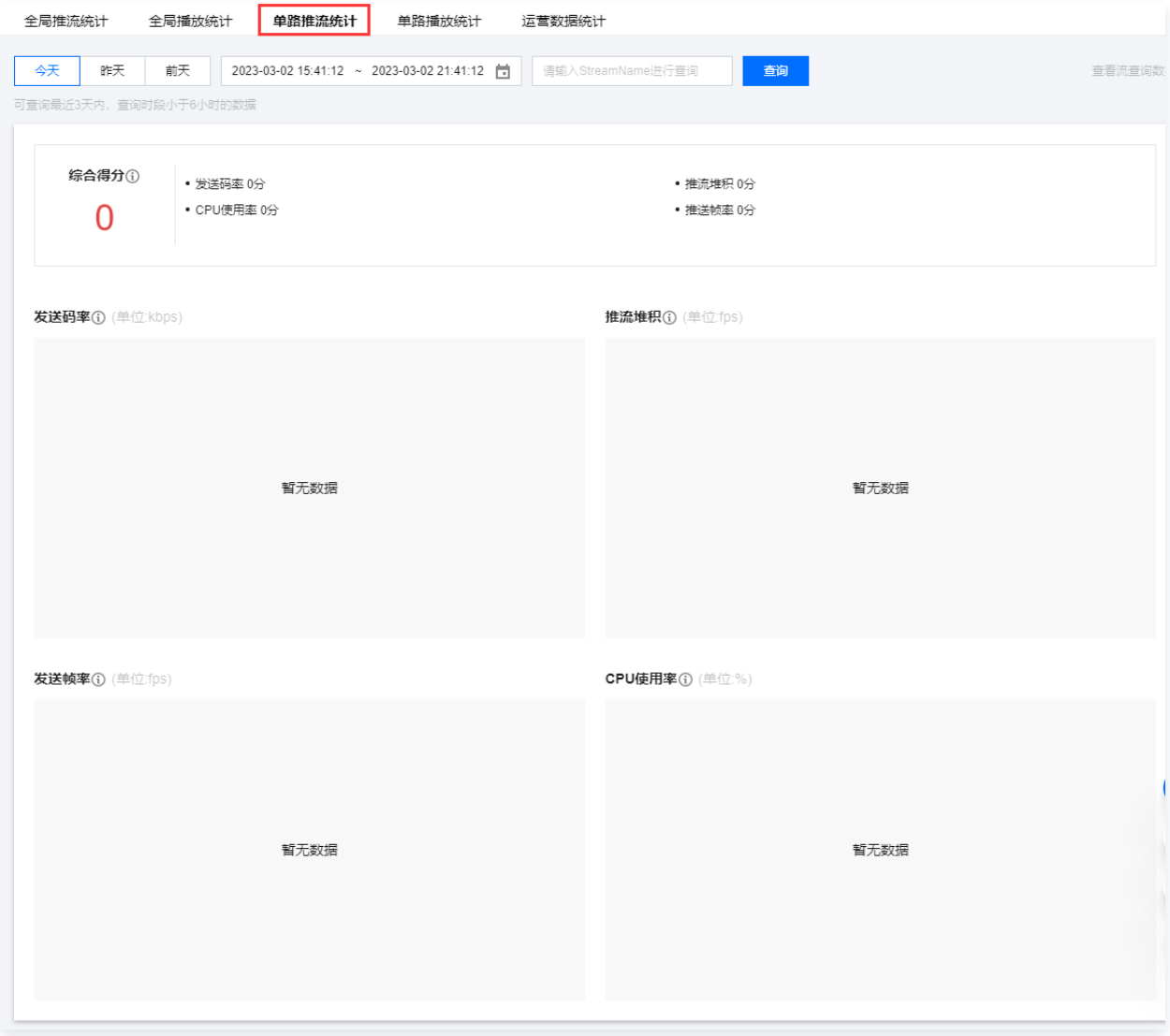
CPU 使用率	播放端 CPU 使用率，分为 App/系统。



单路推流统计

可通过设定开始时间和结束时间，或输入流 ID 进行数据查询。数据统计包括发送码率、推流堆积、发送帧率和 CPU 使用率。

统计项	说明
综合得分	单路流综合得分的均值。
发送码率	视频应发速率与音频应发速率之和与网络实发速率之间的关系。越接近，则主播端推送越流畅，反之主播推送不流畅，易引起播放端观看卡顿。
推流堆积	主播端不能及时发送到云端的音视频数据会被堆积，数据堆积超过三秒（图中红色警示线以上部分）会被主动丢弃。因此当主播的上行网络不理想时，观看端会出现画面的卡顿和明显的跳帧。
发送帧率	当视频推送帧率**低于10FPS**，播放端观看会有看幻灯片的感觉，欠流畅。
CPU 使用率	主播端推流过程中 CPU 占比超过95%，将可能影响主播的推流程序运行的流畅度，可能引起推流卡的情况。当前 Android 8.0 以上的系统 CPU 数据暂时无法获取。



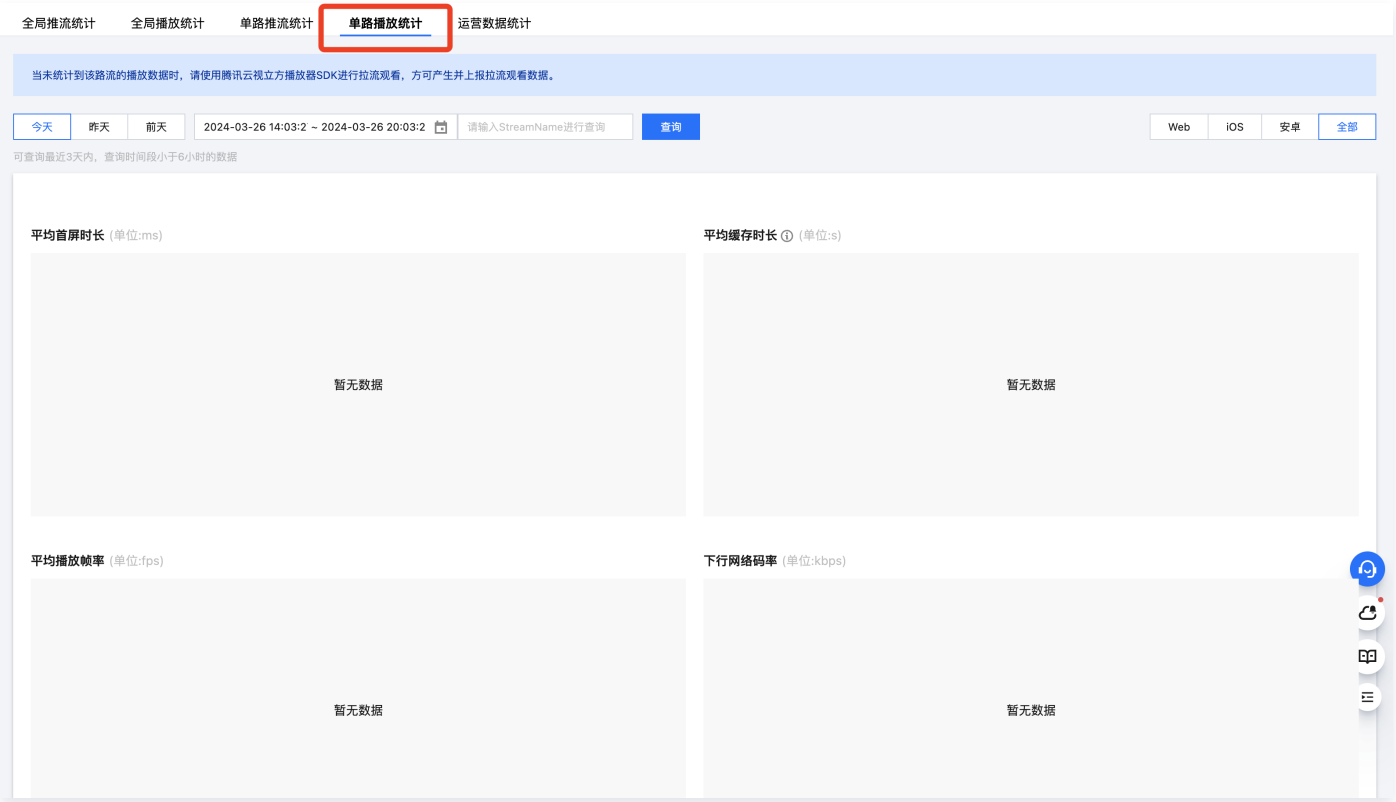
单路播放统计

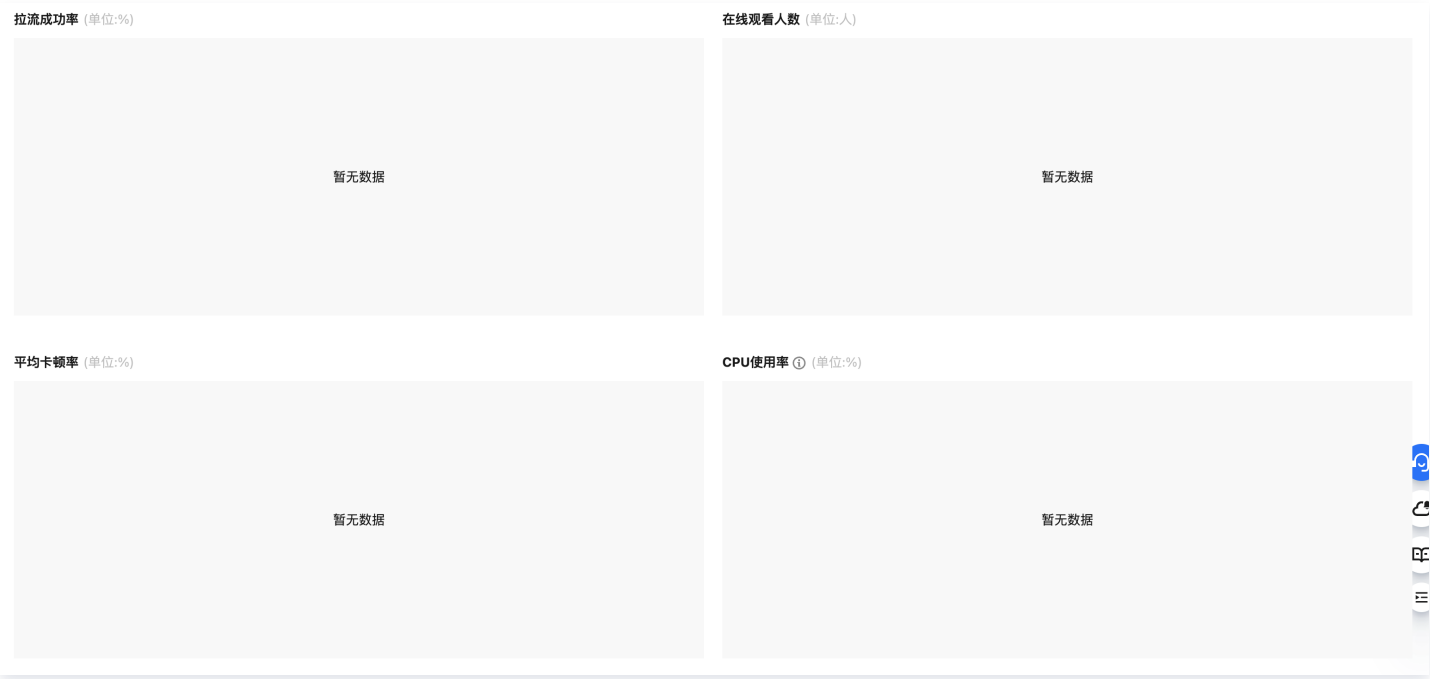
可通过设定开始时间和结束时间，输入流 ID、或选择设备系统进行数据查询。数据统计包括平均首屏时长、平均缓存时长、平均播放帧率、下行网络码率、拉流成功率、在线观看人数、平均卡顿率、CPU 使用率。

统计项	说明
平均首屏时长	不同时刻单路流播放端平均首帧时长。
平均缓存时长	不同时刻单路流播放端平均缓存时长。
平均播放帧率	不同时刻单路流播放端平均播放帧率。
下行网络码率	不同时刻单路流播放端平均下行拉流码率。
拉流成功率	单路流的成功拉流数/单路流的总拉流数。
在线观看人数	不同时刻单路流播放端的拉流数。
平均卡顿率	不同时刻播放端卡顿拉流数 / 总的拉流数。

版权所有：腾讯云计算（北京）有限责任公司

CPU 使用率	播放端 CPU 使用率，分为 App/系统。



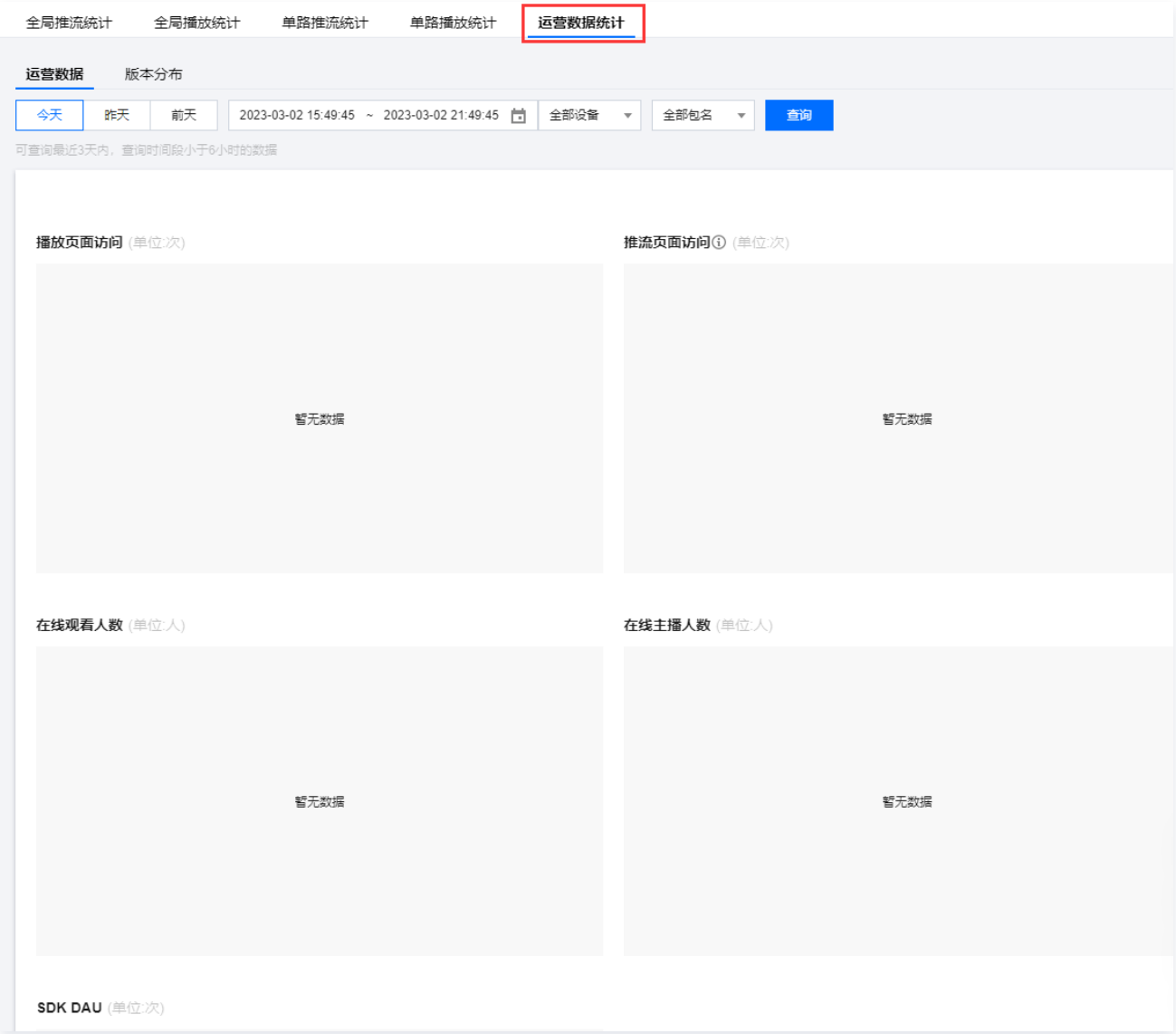


运营数据统计

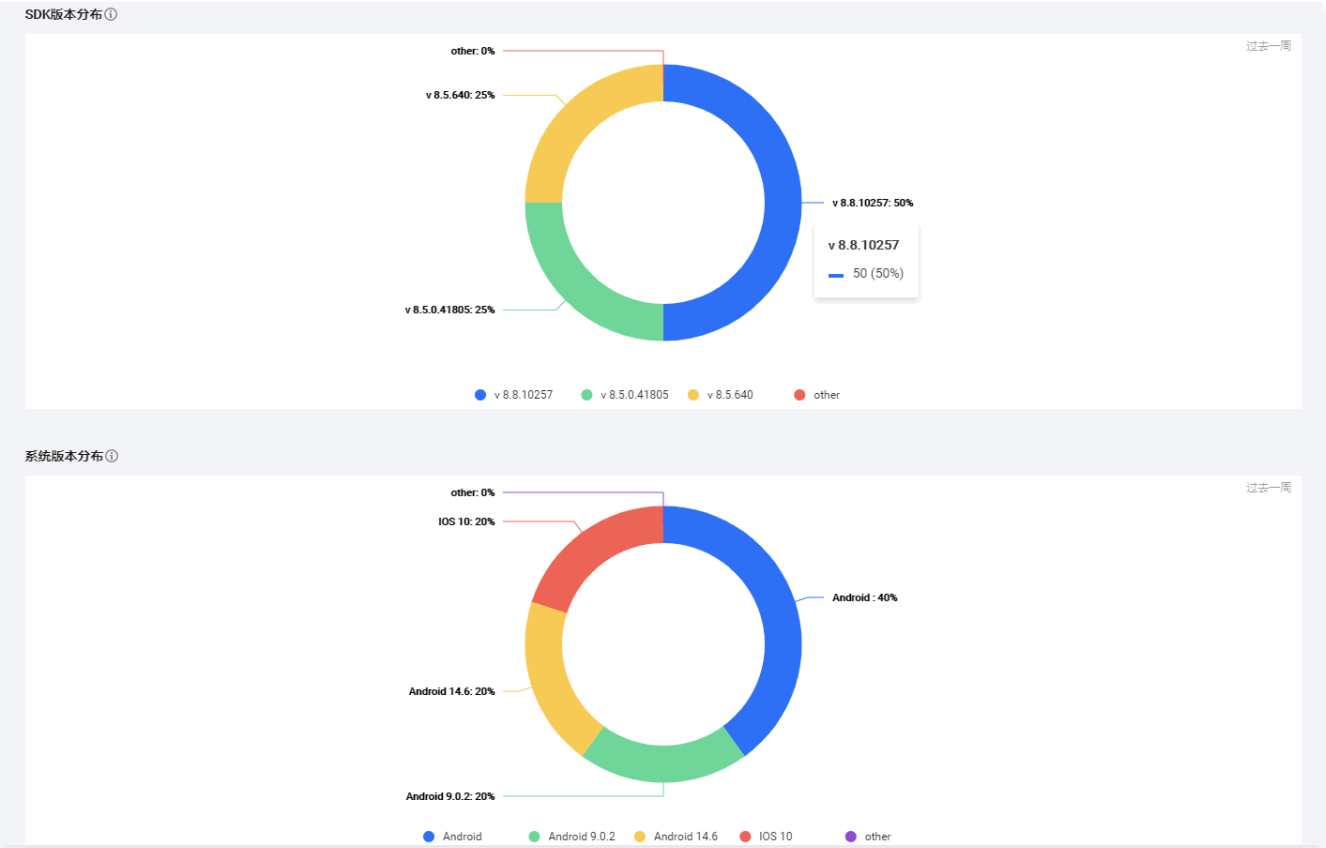
运营数据

可通过设定开始时间和结束时间，或选择设备系统进行数据查询。数据统计包括播放页面访问、推流页面访问、在线观看人数、在线主播人数、SDK DAU。

统计项	说明
播放页面访问	不同时刻不同维度播放端被访问的次数 PV。
推流页面访问	不同时刻不同维度推流端被访问的次数 PV。
在线观看人数	不同时刻不同维度播放端被访问的次数 UV。
在线主播人数	不同时刻不同维度推流端被访问的次数 UV。
SDK DAU	不同时刻不同维度 SDK 的活跃程度 = 播放端 UV + 推流端 UV。



版本分布

统计项	说明
SDK 版本分布	过去一周内，您的用户使用 SDK 的版本分布情况。
系统版本分布	过去一周内，您的用户设备系统的版本分布情况。

设备类型	过去一周内，您的用户设备机型的版本分布情况。