直播 SDK

高级功能

腾讯云

文档目录

高级功能
设定画面质量

最近更新时间：2023-09-19 18:43:53

示例代码

针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

所属平台	GitHub 地址
iOS	Github
Android	Github

功能介绍

LiteAVSDK 通过 V2TXLivePusher 提供的 `setVideoQuality` 接口来设定画面质量：

接口定义

可以通过 `setVideoQuality` 设置推流视频分辨率，以及宽高比模式（横屏 / 竖屏）。

```
public abstract int setVideoQuality(V2TXLiveVideoEncoderParam param);
```

参数

参数	类型	含义
param	V2TXLiveVideoEncoderParam	视频编码参数。

- **V2TXLiveVideoResolution 枚举值：**

取值	含义
V2TXLiveVideoResolution160x160	分辨率 160*160，码率范围：100Kbps ~ 150Kbps，帧率: 15fps
V2TXLiveVideoResolution270x270	分辨率 270*270，码率范围：200Kbps ~ 300Kbps，帧率：15fps
V2TXLiveVideoResolution480x480	分辨率 480*480，码率范围：350Kbps ~ 525Kbps，帧率：15fps
V2TXLiveVideoResolution320x240	分辨率 320*240，码率范围：250Kbps ~ 375Kbps，帧率：15fps。

V2TXLiveVideoResolution480x360	分辨率 480*360，码率范围：400Kbps ~ 600Kbps，帧率：15fps
V2TXLiveVideoResolution640x480	分辨率 640*480，码率范围：600Kbps ~ 900Kbps，帧率：15fps
V2TXLiveVideoResolution320x180	分辨率 320*180，码率范围：250Kbps ~ 400Kbps，帧率：15fps
V2TXLiveVideoResolution480x270	分辨率 480*270，码率范围：350Kbps ~ 550Kbps，帧率：15fps
V2TXLiveVideoResolution640x360	分辨率 640*360，码率范围：500Kbps ~ 900Kbps，帧率：15fps
V2TXLiveVideoResolution960x540	分辨率 960*540，码率范围：800Kbps ~ 1500Kbps，帧率：15fps
V2TXLiveVideoResolution1280x720	分辨率 1280*720，码率范围：1000Kbps ~ 1800Kbps，帧率：15fps
V2TXLiveVideoResolution1920x1080	分辨率 1920*1080，码率范围：2500Kbps ~ 3000Kbps，帧率：15fps

- **V2TXLiveVideoResolutionMode 枚举值：**

取值	含义
V2TXLiveVideoResolutionModeLandscape	横屏模式下的分辨率：V2TXLiveVideoResolution640_360 + V2TXLiveVideoResolutionModeLandscape = 640x360
V2TXLiveVideoResolutionModePortrait	竖屏模式下的分辨率：V2TXLiveVideoResolution640_360 + V2TXLiveVideoResolutionModePortrait = 360x640

参数设定建议

应用场景	resolution	resolutionMode
秀场直播	• V2TXLiveVideoResolution960x540 • V2TXLiveVideoResolution1280x720	横屏或者竖屏
手游直播	V2TXLiveVideoResolution1280x720	横屏或者竖屏
连麦（主画面）	V2TXLiveVideoResolution640x360	横屏或者竖屏
连麦（小画面）	V2TXLiveVideoResolution480x360	横屏或者竖屏

蓝光直播	V2TXLiveVideoResolution1920x1080	横屏或者竖屏

注意事项

为了连麦更流畅，进入连麦状态后请调用 setVideoQuality() 将 quality 挡位设置为 `V2TXLiveVideoResolution640x360`（主播）或 `V2TXLiveVideoResolution480x360`（连麦观众），结束连麦状态后可以调用 `setVideoQuality()` 将 quality 挡位恢复为连麦前的值。

常见问题

1. 为什么观众端看到的画面没有主播端清晰?

主播端看到的画面，是从摄像头采集的原始画面，经过前处理（美颜、镜像、裁剪等操作）后直接渲染给主播观看，所以清晰度是最高的。而观众端看到的是经过编码器压缩再解码的画面，由于编码本身会降低压缩质量（视频码率设置的越低，压缩程度越严重），所以观众端看到的画面会比主播端清晰度低。

2. 为什么 V2TXLivePusher 推出来的流会有368 × 640或者544 × 960这样的分辨率?

在开启硬件加速后，您可能会发现诸如368 × 640或者544 × 960此类“不完美”分辨率，这是由于部分硬编码器要求像素能被16整除所致，属于正常现象，您可以通过设置播放端的填充模式解决“小黑边”问题。

美颜特效
美颜特效

最近更新时间：2023-09-19 18:43:53

腾讯云视立方 · 腾讯特效引擎（Tencent Effect）SDK 是音视频终端 SDK（腾讯云视立方）的重要组成部分，它提供美颜特效功能。基于优图精准的 AI 能力和天天 P 图丰富的实时特效处理，为各类视频处理场景提供丰富的产品能力。腾讯特效 SDK 支持与腾讯云视立方 · 直播 SDK 、短视频 SDK、音视频通话 SDK 等音视频终端产品集成，高效便捷，优势尤为明显。

- 腾讯特效 SDK 功能说明
- 腾讯特效 SDK 集成直播指引

SDK 功能说明

最近更新时间：2024-11-21 10:44:02

腾讯特效 SDK（移动端 / PC 端）能力以套餐和原子能力的形式提供。共有 14 个套餐，分为 3 个系列：A 系列基础套餐 、S 系列高级套餐 和 V 系列虚拟人套餐 。不同系列的不同套餐对应不同功能； 原子能力 提供单算法能力，集成更加灵活，业务拓展性高。
套餐与原子能力支持的功能详情如下表，更多下载说明请参见 SDK 下载 。

> **说明：**
> Web 美颜特效提供的能力以套餐的形式，具体套餐内容可参见 价格说明 ；Web端 SDK 提供 NPM 包和 JS 文件两种接入方式供客户选择，更多详情参考 Web 美颜 SDK 接入 。

A 系列基础套餐功能

A 系列基础套餐提供通用美型功能，适用于对脸部美颜调整要求较低的客户。

套餐功能		套餐编号					
		A1 – 01	A1 – 02	A1 – 03	A1 – 04	A1 – 05	A1 – 06
基础功能	基础美颜美白、磨皮、红润	✓	✓	✓	✓	✓	✓
	画面调整对比度、饱和度、清晰度	✓	✓	✓	✓	✓	✓
	基础美型大眼、瘦脸（自然、女神、英俊）	✓	✓	✓	✓	✓	✓
	滤镜（默认 20 款通用滤镜）	✓	✓	✓	✓	✓	✓
可拓展功能	贴纸（赠送 10 款可选 2D 通用贴纸）	–	✓	✓	✓	✓	✓
	通用美型 SDK（窄脸/下巴/发际线/瘦鼻）	–	–	✓	–	–	–
	手势识别（赠送 1 款指定手势贴纸）	–	–	–	✓	–	–
	人像分割 / 虚拟背景（赠送 3 款指定分割贴纸）	–	–	–	–	✓	–
	美妆（赠送 3 款指定整妆）	–	–	–	–	–	✓
SDK 下载		下载					

S 系列高级套餐功能

S 系列高级套餐提供高级美型功能（包括特效贴纸和美妆），适用于对脸部美颜调整需求较高的客户。

套餐功能		套餐编号					
		S1 – 00	S1 – 01	S1 – 02	S1 – 03	S1 – 04	S1 – 07
基础功能	**基础美颜** 美白、磨皮、红润	✓	✓	✓	✓	✓	✓
	画面调整 对比度、饱和度、清晰度	✓	✓	✓	✓	✓	✓
	高级美型 大眼、窄脸、瘦脸（自然、女神、英俊）、V 脸、下巴、短脸、脸型、发际线、亮眼、眼距、眼角、瘦鼻、鼻翼、瘦颧骨、鼻子位置、白牙、去皱、去法令纹、去眼袋、嘴型、嘴唇厚度、口红、腮红、立体	✓	✓	✓	✓	✓	✓
	滤镜 （默认20款通用滤镜）	✓	✓	✓	✓	✓	✓
	贴纸 （赠送 10 款指定 2D 通用贴纸）	–	✓	✓	✓	✓	✓
	高级贴纸 （赠送 3 款指定 3D 通用贴纸）	–	✓	✓	✓	✓	✓
	美妆 （赠送 3 款指定整妆）	–	✓	✓	✓	✓	✓
可拓展功能	**手势识别** （赠送 1 款指定手势贴纸）	–	–	✓	–	✓	✓
	人像分割 / 虚拟背景 （赠送 3 款指定分割贴纸）	–	–	–	✓	✓	✓
	美形美体 一键瘦身、长腿、瘦腿、瘦肩、小头	–	–	–	–	–	✓
SDK 下载		下载					

V 系列虚拟人套餐

V 系列虚拟人套餐提供虚拟形象 Animoji、形象制作（捏脸）与驱动等功能，适用于虚拟社交、虚拟直播等场景。

套餐功能		套餐编号		
		V1 – 00	V1 – 01	V1 – 02
基础功能	**虚拟形象** 自研 3D 渲染轻量引擎	✓	✓	✓

	捏脸 DIY 支持眼、鼻、嘴、脸型、头发等50+细节维度	✓	–	✓
	面部点位识别与驱动 人脸256关键点识别跟踪、52种面部表情绑定和驱动	–	✓	✓
SDK 下载	iOS & Android	下载		

X 系列原子能力

X 系列提供单独的算法能力，集成更加灵活，业务拓展性更高，适用于对算法能力有需求的客户。

功能	能力编号	
	X1 – 01	X1 – 02
功能名称	人像分割	人脸点位
功能详解	在直播、会议等场景实现虚拟背景，实时精准分割，支持自定义背景	人脸检测（识别人脸出框、多人脸、面部遮挡），256个面部关键点位识别与输出
SDK 下载	下载	

SDK 集成指引
iOS

最近更新时间：2024-08-14 20:00:51

集成准备

1. 下载并解压 Demo 包。
2. 将 Demo工程中的 xmagic 模块（bundle，XmagicIconRes，Xmagic 文件夹）导入到实际项目工程中。
3. **如果使用的 XMagic SDK 版本在2.5.0之前，**导入 SDK 目录中的 `libpag.framework`、`Masonry.framework`、`XMagic.framework`、`YTCommonXMagic.framework`。**如果使用的 XMagic SDK 版本在2.5.1及以后，**导入 SDK 目录中的 `libpag.framework`、`Masonry.framework`、`XMagic.framework`、`YTCommonXMagic.framework`、`Audio2Exp.framework`、`TEFFmpeg.framework`。
4. framework 签名 **General-->Masonry.framework** 和 **libpag.framework** 选 **Embed & Sign**。**YTCommonXMagic.framework** 在版本2.5.1之前选 **Do Not Embed**，在版本2.5.1及以后选 **Embed & Sign**。
5. 将 Bundle ID 修改成与签发的授权一致。

开发者环境要求

- 开发工具 XCode 11 及以上：App Store 或单击 下载地址。
- 建议运行环境：
  - 设备要求：iPhone 5 及以上；iPhone 6 及以下前置摄像头最多支持到 720p，不支持 1080p。
  - 系统要求：iOS 10.0 及以上。

C/C++层开发环境

XCode 默认 C++ 环境。

类型	依赖库
系统依赖库	• Accelerate • AssetsLibrary • AVFoundation • CoreMedia • CoreFoundation • CoreML • Foundation • JavaScriptCore • libc++.tbd • libz.b • libresolv.tbd • libsqlite3.0.tbd • MetalPerformanceShaders • MetalKit

	• MobileCoreServices • OpneAL • OpneGLES • ReplayKit • SystemConfiguration • UIKit
自带的库	• YTCommon（鉴权静态库） • XMagic（美颜静态库） • libpag（视频解码动态库） • Masonry（控件布局库） • TXLiteAVSDK_Professional • TXFFmpeg • TXSoundTouch • Audio2Exp（xmagic sdk version在2.5.1及以后的版本才有） • TEFFmpeg（xmagic sdk version在2.5.1及以后的版本才有）

SDK 接口集成

- 步骤一 和 步骤二 可参考 Demo 工程中，`ThirdBeautyViewController` 类 `viewDidLoad`，`buildBeautySDK` 方法；`AppDelegate` 类的 `application` 方法进行了Xmagic鉴权。
- 步骤四 至 步骤七 可参考 Demo 工程的 `ThirdBeautyViewController`，`BeautyView` 类相关示例代码。

步骤一：初始化授权

1. 首先在工程 `AppDelegate` 的 `didFinishLaunchingWithOptions` 中添加如下鉴权代码，其中 `LicenseURL`，`LicenseKey` 为腾讯云官网申请到的授权信息，请参见 License 指引：

```
[TXLiveBase setLicenceURL:LicenseURL key:LicenseKey];
```

2. Xmagic 鉴权：在相关业务模块的初始化代码中设置 URL 和 KEY，触发 License 下载，避免在使用前才临时去下载。也可以在 `AppDelegate` 的 `didFinishLaunchingWithOptions` 方法里触发下载。其中 `LicenseURL` 和 `LicenseKey` 是控制台绑定 License 时生成的授权信息。SDK版本在2.5.1以前，`TELicenseCheck.h` 在 `XMagic.framework` 里面；SDK版本在2.5.1及以后，`TELicenseCheck.h` 在 `YTCommonXMagic.framework` 里面。

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger
authresult, NSString * _Nonnull errorMsg) {
    if (authresult == TELicenseCheckOk) {
        NSLog(@"鉴权成功");
    } else {
        NSLog(@"鉴权失败");
    }
}];
```

鉴权 errorCode 说明：

错误码	说明
0	成功。Success
-1	输入参数无效，例如 URL 或 KEY 为空
-3	下载环节失败，请检查网络设置
-4	从本地读取的 TE 授权信息为空，可能是 IO 失败引起
-5	读取 VCUBE TEMP License文件内容为空，可能是 IO 失败引起
-6	v_cube.license 文件 JSON 字段不对。请联系腾讯云团队处理
-7	签名校验失败。请联系腾讯云团队处理
-8	解密失败。请联系腾讯云团队处理
-9	TELicense 字段里的 JSON 字段不对。请联系腾讯云团队处理
-10	从网络解析的 TE 授权信息为空。请联系腾讯云团队处理
-11	把TE授权信息写到本地文件时失败，可能是 IO 失败引起
-12	下载失败，解析本地 asset 也失败
-13	鉴权失败
其他	请联系腾讯云团队处理

步骤二：设置 SDK 素材资源路径

```
CGSize previewSize = [self
getPreviewSizeByResolution:self.currentPreviewResolution];
NSString *beautyConfigPath =
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES)
lastObject];
beautyConfigPath = [beautyConfigPath
stringByAppendingPathComponent:@"beauty_config.json"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] &&
!isDir) {
    NSString *beautyConfigJsonStr = [NSString
stringWithContentsOfFile:beautyConfigPath encoding:NSUTF8StringEncoding
error:nil];
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr
dataUsingEncoding:NSUTF8StringEncoding];
```

```
    beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
                                                       options:NSJSONReadingMutableContainers
                                                       error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize
assetsDict:assetsDict];
```

步骤三：添加日志和事件监听

```
    // Register log
    [self.beautyKit registerSDKEventListener:self];
    [self.beautyKit registerLoggerListener:self
withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

步骤四：配置美颜各种效果

```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString
*_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id
_Nullable)extraInfo;
```

步骤五：进行渲染处理

在视频帧回调接口，构造 YTProcessInput 传入到 SDK 内做渲染处理，可参考 Demo 中的 ThirdBeautyViewController。

```
[self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft
withOrientation:YtLightCameraRotation0]
```

步骤六：暂停/恢复 SDK

```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

步骤七：布局中添加 SDK 美颜面板

```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
```

```
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
        gSafeInset = UIEdgeInsetsZero;
    }

dispatch_async(dispatch_get_main_queue(), ^{
    //美颜选项界面
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // 适配全面屏
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
    _vBeauty.hidden = YES;
});
```

Android

最近更新时间：2023-10-11 16:20:22

步骤一：解压 Demo 工程

1. 下载集成了腾讯特效 TE 的 MLVB Demo 工程。本 Demo 基于腾讯特效 SDK S1-04 套餐构建。
2. 替换资源。由于本 Demo 工程使用的 SDK 套餐未必与您实际的套餐一致，因此要将本 Demo 中的相关 SDK 文件替换为您实际使用的套餐的 SDK 文件。具体操作如下：
   - 删除 xmagic 模块中 libs 目录下的 `.aar` 文件，将 SDK 中 libs 目录下的 `.aar` 文件拷贝进 xmagic 模块中 libs 目录下。
   - 删除 xmagic 模块中 assets 目录下的所有文件，将 SDK 中的 `assets/` 目录下的全部资源拷贝到 xmagic 模块 `../src/main/assets` 目录下，如果 SDK 包中的 MotionRes 文件夹内有资源，将此文件夹也拷贝到 `../src/main/assets` 目录下 。
   - 删除 xmagic 模块中 jniLibs 目录下的所有 `.so` 文件，在 SDK 包内的 jniLibs 中找到对应的 .so 文件（由于 SDK 中 jinLibs 文件夹下的 arm64-v8a 和 armeabi-v7a 的 `.so` 文件在压缩包中，所以需要先解压），拷贝到 xmagic 模块中的 `../src/main/jniLibs` 目录下。
3. 将 Demo 程中的 xmagic 模块引 到实际项 程中。

步骤二：打开 app 模块的 build.gradle

1. 将 applicationId 修改成与申请的测试授权 致的包名。
2. 添加 gson 依赖设置。

```
configurations {
all*.exclude group: 'com.google.code.gson'
}
```

步骤三：SDK 接口集成

可参考 Demo 程的 ThirdBeautyActivity 类。

1. **授权：**

```
 //鉴权注意事项及错误码详情，请参考
https://cloud.tencent.com/document/product/616/65891#.E6.AD.A5.E9.AA.A4.E4.B8.80
.EF.BC.9A.E9.89.B4.E6.9D.83
XMagicImpl.checkAuth((errorCode, msg) -> {
        if (errorCode == TELicenseCheck.ERROR_OK) {
            showLoadResourceView();
        } else {
            TXCLog.e(TAG, "鉴权失败，请检查鉴权url和key" + errorCode + " " + msg);
        }
    });
```

2. **初始化素材：**

```
private void showLoadResourceView() {
    if (XmagicLoadAssetsView.isCopyedRes) {
        XmagicResParser.parseRes(getApplicationContext());
        initXMagic();
    } else {
        XmagicLoadAssetsView loadAssetsView = new XmagicLoadAssetsView(this);
        loadAssetsView.setOnAssetsLoadFinishListener(() -> {
            XmagicResParser.parseRes(getApplicationContext());
            initXMagic();
        });
    }
 }
```

3. 启动推流设置：

```
String userId = String.valueOf(new Random().nextInt(10000));
String pushUrl = AddressUtils.generatePushUrl(streamId, userId, 0);
mLivePusher = new V2TXLivePusherImpl(this,
V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
mLivePusher.enableCustomVideoProcess(true, V2TXLivePixelFormatTexture2D,
V2TXLiveBufferTypeTexture);
mLivePusher.setObserver(new V2TXLivePusherObserver() {
 @Override
 public void onGLContextCreated() {
 }

 @Override
 public int onProcessVideoFrame(V2TXLiveDef.V2TXLiveVideoFrame srcFrame,
V2TXLiveDef.V2TXLiveVideoFrame dstFrame) {
    if (mXMagic != null) {
        dstFrame.texture.textureId =
mXMagic.process(srcFrame.texture.textureId, srcFrame.width, srcFrame.height);
    }
    return srcFrame.texture.textureId;
    }

    @Override
    public void onGLContextDestroyed() {
    if (mXMagic != null) {
        mXMagic.onDestroy();
    }
 }
});
mLivePusher.setRenderView(mPushRenderView);
mLivePusher.startCamera(true);
int ret = mLivePusher.startPush(pushUrl);
```

```
mLivePusher.startMicrophone();
```

4. **将 textureId 传入到 SDK 内做渲染处理：**
   在 V2TXLivePusherObserver 接口的
   `onProcessVideoFrame(V2TXLiveDef.V2TXLiveVideoFrame srcFrame, V2TXLiveDef.V2TXLiveVideoFrame dstFrame)`
   方法中添加如下代码。

```
if (mXMagic != null) {
 dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId, srcFrame.width,srcFrame.height);
}
return srcFrame.texture.textureId;
```

5. **暂停/销毁 SDK：**
   `onPause()` 用于暂停美颜效果，可以在 Activity/Fragment 生命周期方法中执行， onDestroy 方法需要在 GL 线程调用（可以在 onTextureDestroyed 方法中调用 XMagicImpl 对象的 `onDestroy()` ），更多使用请参考 Demo。

```
mXMagic.onPause();    //暂停，与Activity的onPause方法绑定
mXMagic.onDestroy();   // //销毁，需要在GL线程中调用
```

6. **布局中添加 SDK 美颜面板：**

```
<RelativeLayout
    android:id="@+id/livepusher_bp_beauty_pannel"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@+id/ll_edit_info" />
```

7. **初始化面板：**

```
 private void initXMagic() {
      if (mXMagic == null) {
          mXMagic = new XMagicImpl(this, mBeautyPanelView);
      }else {
          mXMagic.onResume();
      }
  }
```

具体操作请参见 Demo 程的 `ThirdBeautyActivity.initXMagic();` 法。

HLS 自适应码率播放

最近更新时间：2025-01-21 10:42:32

示例代码

针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

所属平台	GitHub 地址
iOS	Github
Android	Github

功能介绍

在了解自适应码率播放之前，首先要了解什么是无缝切流。

- **无缝切流**：在切换不同码率的转码流时，能够做到无缝衔接，不会出现声音画面中断或者跳变的情况，实现观感和听感的平滑过渡。
- **自适应码率播放**：在无缝切流的基础上更进一步，无需用户干预，完全由当前网络带宽来决定实现自动无缝切流，减少网络波动对播放流畅度的影响。

开启自适应播放

自适应播放功能依托于腾讯云的直播自适应码率支撑，如果您想要对接这个功能，需要在腾讯云的管理控制台 开通直播自适应码率服务 。服务开通之后，需要创建至少一个自适应模板并包含两个子流，才可使用自适应播放功能。

> **说明：**
> - 自适应播放会在创建的自适应模板中，通过当前网络带宽来选择一个子流进行播放。
> - 在使用自适应码率功能时，会产生模板中所有子流的 转码费用 。

创建自适应码率模板

1. 登录 云直播控制台 ，选择**功能配置** > **直播自适应码率**，创建自适应码率模板。如下图所示：

2. **配置子流：**至少应该配置两个子流。

3. **绑定域名**：创建转码模板之后，需要绑定域名。

说明：
配置自适应码率模板的详细参数，请参见 直播自适应码率 。

获取 HLS 自适应播放地址

1. 通过 地址生成器，选择创建的自适应码率模板，单击生成。

2. 通过自适应模板的名称，来拼接地址。HLS 自适应播放的格式为：

```
http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3u8
```

在上述的 URL 中，存在一些关键字段，关于其中关键字段的含义信息，详见下表：

字段名称	字段含义

http://	HLS URL 的前缀字段
Domain	HLS 直播播放域名
AppName	应用名称，指的是直播流媒体文件存放路径，默认云直播会分配一个路径：live
StreamName	流名称，指每路直播流唯一的标识符
AdaptiveTemplate	自适应码率模板名称，表明使用哪一个配置模板

URL 地址样例（hlsAutoTest 为自适应码率模板名称）：

类型	URL地址
原始	http://xxx.liveplay.myqcloud.com/stream_name.m3u8
自适应播放	http://xxx.liveplay.myqcloud.com/stream_name_hlsAutoTest.m3u8

实现 HLS 自适应播放

使用 V2TXLivePlayer 对象可以使用 HLS 自适应播放，具体做法如下（传入正确的 URL 是关键）：
示例代码：

Android

```
// 创建□个 V2TXLivePlayer 对象;
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传□HLS自适应播放地址，即可开始播放;
player.startLivePlay("http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3
u8");
```

iOS

```
// 创建□个 V2TXLivePlayer 对象;
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
[player setObserver:self];
[player setRenderView:videoView];
// 传□HLS自适应播放地址，即可开始播放;
[player
startLivePlay:@"http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3u8"];
```

实现 HLS 无缝切流

使用 V2TXLivePlayer 对象可以使用 HLS 无缝切流，具体做法如下：
1. 播放自适应地址，并且设置 Observer 回调。

Android

```
// 创建一个 V2TXLivePlayer 对象;
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传入HLS自适应播放地址，即可开始播放;
player.startLivePlay("http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3
u8");
```

iOS

```
// 创建一个 V2TXLivePlayer 对象;
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
[player setObserver:self];
[player setRenderView:videoView];
// 传入HLS自适应播放地址，即可开始播放;
[player
startLivePlay:@"http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3u8"];
```

2. 在 onConnected 回调中，获取全部的子流地址。

Android

```
/**
 * 已经成功连接到服务器
 * @param player     回调该通知的播放器对象
 * @param extraInfo 扩展信息
 */
public void onConnected(V2TXLivePlayer player, Bundle extraInfo){
    ArrayList<V2TXLiveStreamInfo> streams = player.getStreamList();
}
```

iOS

```
/**
 * 已经成功连接到服务器
 * @param player     回调该通知的播放器对象
 * @param extraInfo 扩展信息
 */
- (void)onConnected:(id<V2TXLivePlayer>)player extraInfo:(NSDictionary *)extraInfo
{
    self.streams = [player getStreamList]
}
```

3. 切换到一个子流

Android

```
// 选择期望质量的码流
V2TXLiveStreamInfo stream = streams.get(index);
// 切换到期望质量的码流
player.switchStream(stream.url);
```

iOS

```
// 选择期望质量的码流
V2TXLiveStreamInfo *streamInfo = self.streams[index];
// 切换到期望质量的码流
[self.player switchStream:streamInfo.url];
```

4. 切换回自适应播放

Android

```
player.switchStream("http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3u
8");
```

iOS

```
[self.player
switchStream:@"http://{Domain}/{AppName}/{StreamName}_{AdaptiveTemplate}.m3u8"];
```

快直播自适应码率播放

最近更新时间：2025-01-21 10:42:32

示例代码

针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

所属平台	GitHub 地址
iOS	Github
Android	Github

功能介绍

在了解自适应码率播放之前，首先要了解什么是无缝切流。

- **无缝切流：**在切换不同码率的转码流时，能够做到无缝衔接，不会出现声音画面中断或者跳变的情况，实现观感和听感的平滑过渡。
- **自适应码率播放：**在无缝切流的基础上更进一步，无需用户干预，完全由当前网络带宽来决定实现自动无缝切流，确保播放的流畅度不受网络波动的影响。

开启自适应播放

自适应播放功能依托于腾讯云的直播自适应码率支撑，如果您想要对接这个功能，需要在腾讯云的管理控制台 开通直播自适应码率服务 。服务开通之后，需要创建至少一个自适应模板并包含两个子流，才可使用自适应播放功能。

> **说明：**
> - 自适应播放会在创建的自适应模板中，通过当前网络带宽来选择一个子流进行播放。
> - 在使用自适应播放功能时，会产生模板中所有子流的 转码费用 。

创建自适应码率模板

1. 登录 云直播控制台 ，选择 **功能配置** > **直播自适应码率**，点击**创建模板**。如下图所示：

2. **配置子流**：至少应该配置两个子流。

3. **绑定域名：**创建转码模板之后，需要绑定域名。

说明：
配置自适应码率模板的详细参数，请参见 直播自适应码率 。

获取快直播自适应播放地址

1. 通过 地址生成器 ，选择创建的自适应码率模板，单击**生成地址**。

2. 通过自适应码率模板中，子流的名称，来拼接地址。

   快直播自适应播放的格式为：

```
webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate={startStream}&tabr_control=auto
```

   在上述的 URL 中，存在一些关键字段，关于其中关键字段的含义信息，详见下表：

字段名称	字段含义
webrtc://	快直播 URL 的前缀字段
Domain	快直播播放域名
AppName	应用名称，指的是直播流媒体文件存放路径，默认云直播会分配一个路径：live
StreamName	流名称，指每路直播流唯一的标识符
tabr_bitrates	转码模板列表，多个子流之间使用逗号隔开。转码模板必须已经在控制台中创建，并且按照码率从高到低进行排列
subStreams	子流名称，多个子流用逗号分隔
tabr_start_bitrate	当前要播放的子流，取值必须是子流列表中的一个
startStream	当前要播放的子流名称
tabr_control	打开自适应码率播放。取值固定为 “auto”，如果是手动无缝切流则不需要

URL 地址样例（1080p，720p, 480p均为子流名称）：

类型	URL地址
原始	webrtc://xxx.liveplay.myqcloud.com/stream_name
无缝切流	webrtc://xxx.liveplay.myqcloud.com/stream_name?tabr_bitrates=720p,480p&tabr_start_bitrate=720p
自适应播放	webrtc://xxx.liveplay.myqcloud.com/stream_name?tabr_bitrates=720p,480p&tabr_start_bitrate=720p&tabr_control=auto

实现快直播自适应码率播放

使用 V2TXLivePlayer 对象可以使用快直播自适应播放，具体做法如下（传入正确的 URL 是关键）：
示例代码：

Android

```
// 创建个 V2TXLivePlayer 对象;
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传快直播自适应播放地址，即可开始播放;
player.startLivePlay("webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate={startStream}&tabr_control=auto");
```

iOS

```
// 创建一个 V2TXLivePlayer 对象;
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
[player setObserver:self];
[player setRenderView:videoView];
// 传入快直播自适应播放地址，即可开始播放;
[player startLivePlay:@"webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate={startStream}&tabr_control=auto"];
```

注意：

- 在设置的 tabr_bitrates 参数中，子流必须按照码率由高到低进行排列。
- 进入自适应码率播放后，将无法进行无缝切流。除非去掉 `tabr_control=auto` 重新 startPlay。

实现快直播无缝切流

使用 V2TXLivePlayer 对象可以使用快直播自适应播放，具体做法如下（传入正确的 URL 是关键）：
示例代码：

Android

```
// 创建一个 V2TXLivePlayer 对象;
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传入无缝切流地址，使用720p开始播放，假设对应子流名称为Auto720p
player.startLivePlay("webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate=Auto720p");
// ...
// 切流到480p，假设对应子流名称为Auto480p
player.switchStream("webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate=Auto480p");
```

iOS

```
// 创建一个 V2TXLivePlayer 对象;
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
[player setObserver:self];
[player setRenderView:videoView];
// 传入无缝切流地址，使用720p开始播放，假设对应子流名称为Auto720p
[player startLivePlay:@"webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate=Auto720p"];
// ...
// 切流到480p，假设对应子流名称为Auto480p
```

```
[player switchStream:@"webrtc://{Domain}/{AppName}/{StreamName}?tabr_bitrates=
{subStreams}&tabr_start_bitrate=Auto480p"];
```

自定义采集和渲染

最近更新时间：2023-09-19 18:43:54

示例代码

针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

所属平台	GitHub 地址
iOS	Github
Android	Github

定制推流画面

iOS 平台

方案一：修改 OpenGL 纹理

如果您有自定义图像处理的需求（例如：堆加字幕），同时又希望复用 LiteAV SDK 的整体流程，您可以按照如下攻略进行定制。

1. 首先需要调用 V2TXLivePusher 的 enableCustomVideoProcess 开启自定义视频处理，才会收到这个回调通知。
2. 处理图像时分为两种情况。
   - **美颜组件会产生新的纹理：**
     如果您使用的美颜组件会在处理图像的过程中产生一帧全新的纹理（用于承载处理后的图像），那请您在回调函数中将 `dstFrame.textureId` 设置为新纹理的 ID。

```
- (void) onProcessVideoFrame:(V2TXLiveVideoFrame _Nonnull)srcFrame
dstFrame:(V2TXLiveVideoFrame _Nonnull)dstFrame
    {
        GLuint dstTextureId = renderItemWithTexture(srcFrame.textureId,
srcFrame.width, srcFrame.height);
        dstFrame.textureId = dstTextureId;
    }
```

   - **美颜组件并不自身产生新纹理：**
     如果您使用的第三方美颜模块并不生成新的纹理，而是需要您设置给该模块一个输入纹理和一个输出纹理，则可以考虑如下方案：

```
- (void) onProcessVideoFrame:(V2TXLiveVideoFrame _Nonnull)srcFrame
dstFrame:(V2TXLiveVideoFrame _Nonnull)dstFrame
    {
```

```
        thirdparty_process(srcFrame.textureId, srcFrame.width,
srcFrame.height, dstFrame.textureId);
    }
```

3. 最后，对于 texture 数据的操作，需要一定的 OpenGL 基础知识，另外计算量不宜太大，因为 onProcessVideoFrame 的调用频率跟 FPS 相同，过于繁重的处理很容易造成 GPU 过热。

方案二：自己采集数据

如果您只希望使用 SDK 来编码和推流（例如已经对接了商汤等产品），视频采集和预处理（即美颜、滤镜这些）全部由自己的代码来控制，可以按如下步骤实现：

1. 调用 V2TXLivePusher 的 enableCustomVideoCapture 接口开启自定义采集。
   这样 SDK 本身就不会再采集视频数据，而只是启动编码、流控、发送等跟推流相关的工作。
2. 通过 V2TXLivePusher 的 sendCustomVideoFrame 向 SDK 填充 Video 数据。

```
/**
 * @brief 在自定义视频采集模式下，将采集的视频数据发送到SDK。
 *        在自定义视频采集模式下，SDK不再采集摄像头数据，仅保留编码和发送功能。
 *        您可以把采集到的 SampleBuffer 打包到 V2TXLiveVideoFrame 中，然后通过该API定
期的发送。
 *
 * @note  需要在 [startPush](@ref V2TXLivePusher#startPush:) 之前调用
[enableCustomVideoCapture](@ref V2TXLivePusher#enableCustomVideoCapture:) 开启
自定义采集。
 *
 * @param videoFrame 向 SDK 发送的 视频帧数据 {@link V2TXLiveVideoFrame}
 *
 * @return 返回值 {@link V2TXLiveCode}
 *         - V2TXLIVE_OK: 成功
 *         - V2TXLIVE_ERROR_INVALID_PARAMETER: 发送失败，视频帧数据不合法
 *         - V2TXLIVE_ERROR_REFUSED: 您必须先调用 enableCustomVideoCapture 开启自
定义视频采集。
 */
- (V2TXLiveCode)sendCustomVideoFrame:(V2TXLiveVideoFrame *)videoFrame;
```

Android 平台

方案一：修改 OpenGL 纹理

如果您有自定义图像处理的需求（例如堆加字幕），同时又希望复用 LiteAV SDK 的整体流程，您可以按照如下攻略进行定制。

1. 首先需要调用 V2TXLivePusher 的 enableCustomVideoProcess 开启自定义视频处理，才会收到这个回调通知。

2. 处理图像时分为两种情况。
   - **美颜组件会产生新的纹理：**
     如果您使用的美颜组件会在处理图像的过程中产生一帧全新的纹理（用于承载处理后的图像），那请您在回调函数中将 `dstFrame.textureId` 设置为新纹理的 ID。

```
private class MyPusherObserver extends V2TXLivePusherObserver {
 @Override
 public void onGLContextCreated() {
 mFURenderer.onSurfaceCreated();
 mFURenderer.setUseTexAsync(true);
 }

 @Override
 public int onProcessVideoFrame(V2TXLiveVideoFrame srcFrame,
V2TXLiveVideoFrame dstFrame) {
 dstFrame.texture.textureId = mFURenderer.onDrawFrameSingleInput(
         srcFrame.texture.textureId, srcFrame.width, srcFrame.height);
 return super.onProcessVideoFrame(srcFrame, dstFrame);;
 }

 @Override
 public void onGLContextDestroyed() {
 mFURenderer.onSurfaceDestroyed();
 }
}
```

   - **美颜组件并不自身产生新纹理：**
     如果您使用的第三方美颜模块并不生成新的纹理，而是需要您设置给该模块一个输入纹理和一个输出纹理，则可以考虑如下方案：

```
 @Override
 public int onProcessVideoFrame(V2TXLiveVideoFrame srcFrame,
V2TXLiveVideoFrame dstFrame) {
 thirdparty_process(srcFrame.texture.textureId, srcFrame.width,
srcFrame.height, dstFrame.texture.textureId);
 return 0;
 }
```

3. 最后，对于 texture 数据的操作，需要一定的 OpenGL 基础知识，另外计算量不宜太大，因为 onProcessVideoFrame 的调用频率跟 FPS 相同，过于繁重的处理很容易造成 GPU 过热。

方案二：自己采集数据

如果您只希望使用 SDK 来编码和推流（例如已经对接了商汤等产品），视频采集预处理（即美颜、滤镜这些）全部由自己的代码来控制，可以按如下步骤实现：

1. 调用 V2TXLivePusher 的 enableCustomVideoCapture 接口开启自定义采集。
   这样 SDK 本身就不会再采集视频数据，而只是启动编码、流控、发送等跟推流相关的工作。
2. 通过 V2TXLivePusher 的 sendCustomVideoFrame 向 SDK 填充 Video 数据。

```
/**
 * @brief 在自定义视频采集模式下，将采集的视频数据发送到SDK。 <br/>
 *        在自定义视频采集模式下，SDK不再采集摄像头数据，仅保留编码和发送功能。
 *
 * @note  需要在 {@link V2TXLivePusher#startPush(String)} 之前调用 {@link
V2TXLivePusher#enableCustomVideoCapture(boolean)} 开启自定义采集。
 *
 * @param videoFrame 向 SDK 发送的 视频帧数据 {@link V2TXLiveVideoFrame}
 *
 * @return 返回值 {@link V2TXLiveCode}
 *         - V2TXLIVE_OK: 成功
 *         - V2TXLIVE_ERROR_INVALID_PARAMETER: 发送失败，视频帧数据不合法
 *         - V2TXLIVE_ERROR_REFUSED: 发送失败，您必须先调用
enableCustomVideoCapture 开启自定义视频采集
 */
public abstract int sendCustomVideoFrame(V2TXLiveVideoFrame videoFrame);
```

定制播放数据

iOS 平台

1. 设置 V2TXLivePlayer 的 V2TXLivePlayerObserver 监听。

```
@interface V2TXLivePlayer : NSObject
/**
 * @brief 设置播放器回调。<br/>
 *        通过设置回调，可以监听 V2TXLivePlayer 播放器的一些回调事件，
 *        包括播放器状态、播放音量回调、音视频首帧回调、统计数据、警告和错误信息等。
 *
 * @param observer 播放器的回调目标对象，更多信息请查看 {@link V2TXLivePlayerObserver}
 */
- (void)setObserver:(id<V2TXLivePlayerObserver>)observer;
```

2. 通过 onRenderVideoFrame 回调捕获 Player 的图像数据。

```
/**
 * @brief 视频帧信息。
 *        V2TXLiveVideoFrame 用来描述一帧视频画面的裸数据，它可以是一帧编码前的画面，也可
以是一帧解码后的画面。
```

```
 * @note  自定义采集和自定义渲染时使用。自定义采集时，需要使用 V2TXLiveVideoFrame 来包装待发送的视频帧；自定义渲染时，会返回经过 V2TXLiveVideoFrame 包装的视频帧。
 */
@interface V2TXLiveVideoFrame : NSObject

///【字段含义】视频帧像素格式
///【推荐取值】V2TXLivePixelFormatNV12
@property(nonatomic, assign) V2TXLivePixelFormat pixelFormat;

///【字段含义】视频数据包装格式
///【推荐取值】V2TXLiveBufferTypePixelBuffer
@property(nonatomic, assign) V2TXLiveBufferType bufferType;

///【字段含义】bufferType 为 V2TXLiveBufferTypeNSData 时的视频数据
@property(nonatomic, strong, nullable) NSData *data;

///【字段含义】bufferType 为 V2TXLiveBufferTypePixelBuffer 时的视频数据
@property(nonatomic, assign, nullable) CVPixelBufferRef pixelBuffer;

///【字段含义】视频宽度
@property(nonatomic, assign) NSUInteger width;

///【字段含义】视频高度
@property(nonatomic, assign) NSUInteger height;

///【字段含义】视频帧的顺时针旋转角度
@property(nonatomic, assign) V2TXLiveRotation rotation;

///【字段含义】视频纹理ID
@property (nonatomic, assign) GLuint textureId;

@end


@protocol V2TXLivePlayerObserver <NSObject>
@optional
/**
 * @brief 自定义视频渲染回调
 *
 * @note  调用 [enableCustomRendering](@ref V2TXLivePlayer#enableCustomRendering:pixelFormat:bufferType:) 开启自定义渲染之后，会收到这个回调通知
 *
 * @param videoFrame  视频帧数据 {@link V2TXLiveVideoFrame}
 */
- (void)onRenderVideoFrame:(id<V2TXLivePlayer>)player
                     frame:(V2TXLiveVideoFrame *)videoFrame;
@end
```

Android 平台

1. 设置 V2TXLivePlayer 的 V2TXLivePlayerObserver 监听。

```
public abstract void setObserver(V2TXLivePlayerObserver observer)
```

2. 通过 onRenderVideoFrame 回调捕获 Player 的图像数据。

```
public final static class V2TXLiveVideoFrame
{
    /// 视频像素格式
    public V2TXLivePixelFormat pixelFormat =
V2TXLivePixelFormat.V2TXLivePixelFormatUnknown;
    /// 视频数据包装格式
    public V2TXLiveBufferType bufferType    =
V2TXLiveBufferType.V2TXLiveBufferTypeUnknown;
    /// 视频纹理包装类
    public V2TXLiveTexture texture;
    /// 视频数据
    public byte[]       data;
    /// 视频数据
    public ByteBuffer   buffer;
    /// 视频宽度
    public int          width;
    /// 视频高度
    public int          height;
    /// 视频像素的顺时针旋转角度
    public int          rotation;
}

public abstract class V2TXLivePlayerObserver {
    /**
     * 自定义视频渲染回调
     *
     *  @param player      回调该通知的播放器对象
     *  @param videoFrame 视频帧数据 {@link V2TXLiveVideoFrame}
     */
    void onRenderVideoFrame(V2TXLivePlayer player, V2TXLiveVideoFrame
videoFrame);
}
```

SDK 指标监控

最近更新时间：2024-08-22 17:24:41

SDK 的各项监控指标可以从 V2TXLivePusherObserver 和 V2TXLivePlayerObserver 的回调中获取。

V2TXLivePusherObserver

获取推流的状态数据

V2TXLivePusherObserver 的 onStatisticsUpdate 回调，会每隔2秒会将 SDK 内部的状态指标同步出来，其中如下指标比较有意义：

推流状态	含义说明
appCpu	当前进程的 CPU 使用率。
systemCpu	本机总体的 CPU 使用率。
width	当前视频的宽度，单位：像素值。
height	当前视频的高度，单位：像素值。
fps	当前视频帧率，也就是视频编码器每秒生产了多少帧画面。
videoBitrate	当前视频编码器输出的比特率，也就是编码器每秒生产了多少视频数据，单位：kbps。
audioBitrate	当前音频编码器输出的比特率，也就是编码器每秒生产了多少音频数据，单位：kbps。

有参考价值的状态指标

状态指标	说明
systemCpu	• 如果**系统 CPU 使用率**超过80%，音视频编码的稳定性会受到影响，可能导致画面和声音的随机卡顿。 • 如果**系统 CPU 使用率**经常100%，会导致视频编码帧率不足，音频编码跟不上，必然导致画面和声音的严重卡顿。
fps	通常来说每秒15帧以上的视频流才能保证观看的流畅度，常规推流如果 FPS 在10帧以下，观众就会明显的感到画面卡顿。

> **说明**
> 很多客户会遇到的一个问题： App 在线下测试时性能表现极佳，但在 App 外发上线后，前排房间里的互动消息的滚屏和刷新会产生极大的 CPU 消耗导致直播画面卡顿严重。

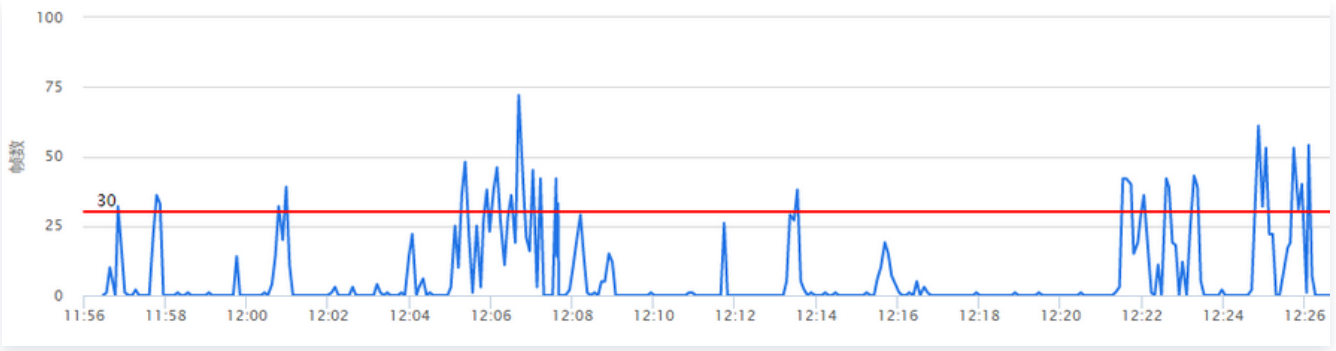
看懂腾讯云推流图表

在**云直播控制台**的 质量监控 您可以看到您所属账户里的直播间情况，以及每个直播间的推流质量数据：

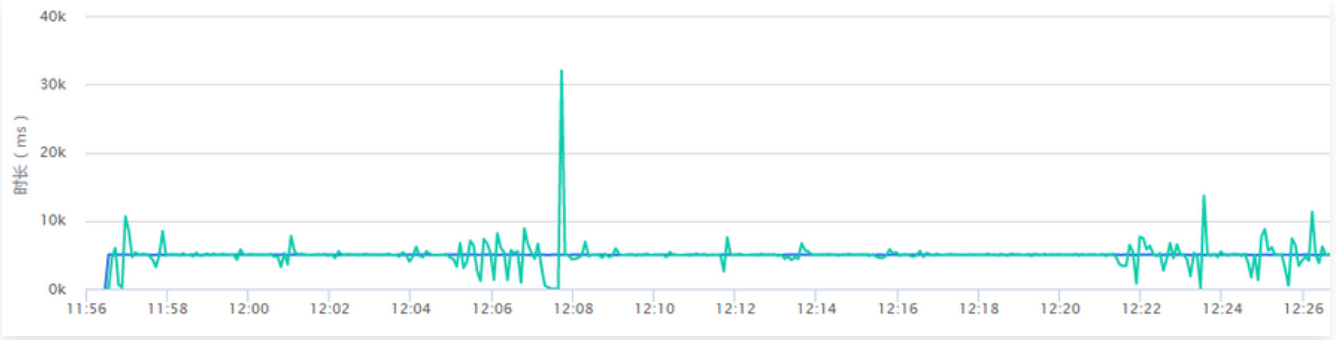
- **主播端-应发速率-实发速率曲线图**
  蓝色曲线代表 BITRATE 的统计曲线，即 SDK 产生的音视频数据，绿色曲线代表实际网络发出去多少。两条线重合度越高表

示推流质量越好。

- **主播端-音视频数据堆积情况**
- 如果曲线始终贴着0刻度线走，说明整个推流过程非常顺畅，一点都没有堆积。
- 如果出现大于0的部分，说明当时有网络波动导致数据积压，有可能在播放端产生轻微卡顿和音画不同步的现象。
- 如果堆积超出红色警戒线，说明已经产生了丢包，必然会在播放端产生卡顿和音画不同步的现象。



- **云端-应收视频时长-实收视频时长曲线**
  这里是腾讯云服务端的统计图表，如果您不是使用腾讯云 SDK 推流，那么您将只能看到这个图表，前面两个（数据源来自 SDK）是看不到的。蓝绿两条线重合度越高，说明推流质量越好。



V2TXLivePlayerObserver

获取播放的状态数据

V2TXLivePlayerObserver 的 onStatisticsUpdate 回调，会每隔2秒将 SDK 内部的状态指标同步出来，其中如下指标比较有意义：

播放状态	含义说明

appCpu	当前进程的 CPU 使用率。
systemCpu	本机总体的 CPU 使用率。
width	当前视频的宽度，单位：像素值。
height	当前视频的高度，单位：像素值。
fps	当前流媒体的视频帧率。
videoBitrate	当前流媒体的视频码率，单位：kbps。
audioBitrate	当前流媒体的音频码率，单位：kbps。

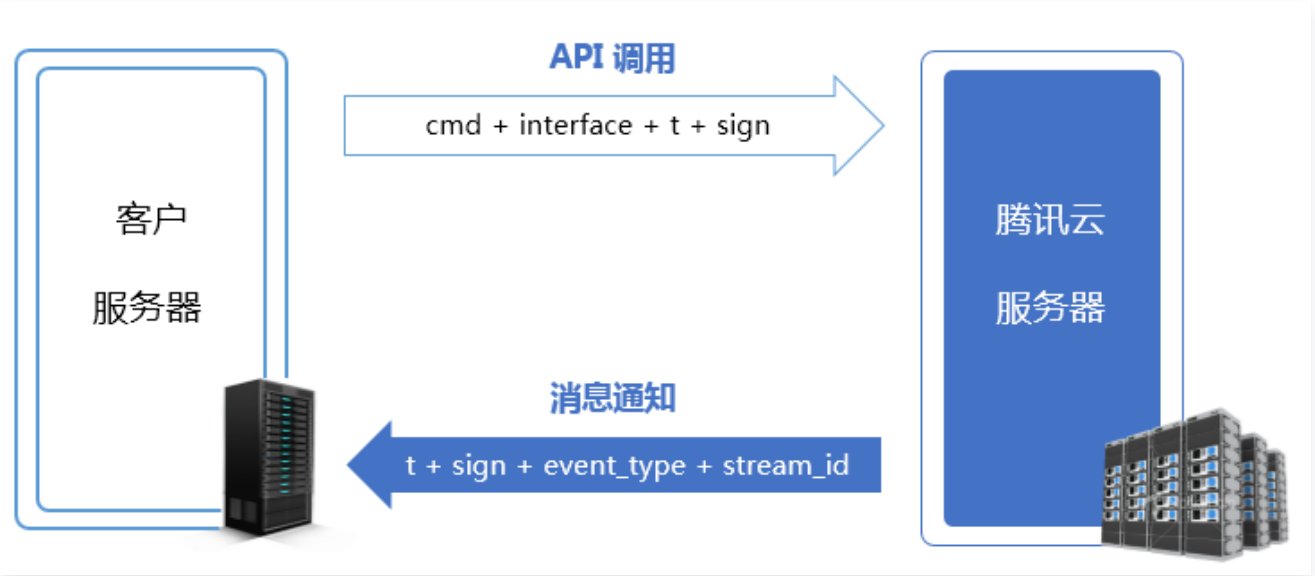
有参考价值的状态指标

状态指标	说明
systemCpu	• 如果**系统 CPU 使用率**超过80%，音视频编码的稳定性会受到影响，可能导致画面和声音的随机卡顿。 • 如果**系统 CPU 使用率**经常100%，会导致视频编码帧率不足，音频编码跟不上，必然导致画面和声音的严重卡顿。
fps	通常来说每秒15帧以上的视频流才能保证观看的流畅度，常规推流如果 FPS 在10帧以下，观众就会明显的感到画面卡顿。

禁播和流管理

最近更新时间：2023-06-16 11:33:22

与腾讯云后台通讯



您的服务器与腾讯云服务器的信息同步可以通过两种方式组合实现：

- **API 调用**：腾讯云提供了直播相关的 API 接口，包括状态查询和状态管理等功能，供您的后台服务器调用。
- **消息通知**：腾讯云在直播流状态变更、录制文件生成等一系列事件发生时，能够以事件消息（JSON）的形式主动通知您的后台服务器，只需要您在腾讯云注册接收事件通知的回调 URL 即可实现。

API 调用

腾讯云提供了直播相关的 API 接口，包括状态查询和状态管理等功能，供您的后台服务器调用，详情请参见 云直播 API 概览 文档。

调用方法

在您的**服务端**采用 HTTP 协议的 GET 请求方式（即调用参数直接拼接在 URL 中）进行调用即可，详细的调用方法在每个 API 的说明文档中都有示例参考。

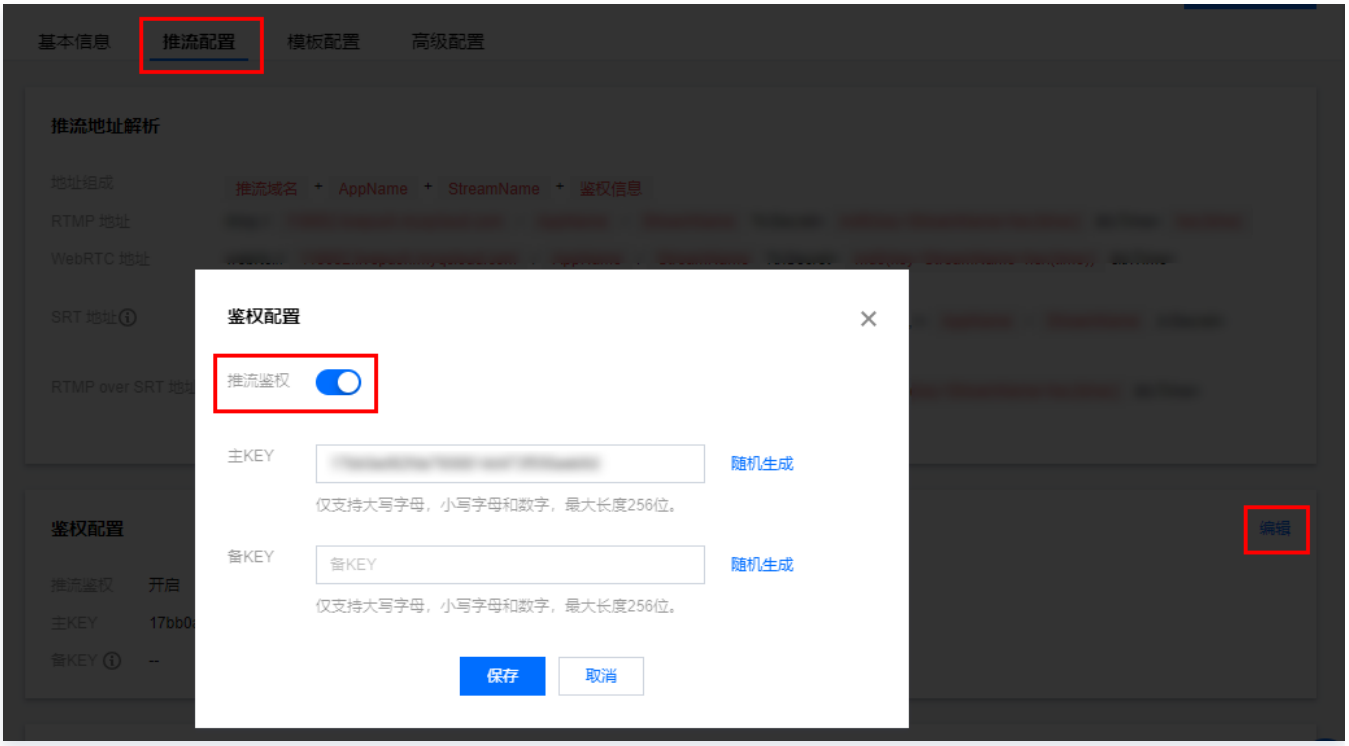
安全机制

由于对 API 的调用采用的是普通的 HTTP 协议（出于性能考虑），这就需要一套行之有效的办法来确保您的服务器与腾讯云后台之间的通讯安全。
所有直播码相关的云端 API 都采用了同一种安全检查机制，**t + sign 校验**：

- **t（过期时间）**：如果一个 API 请求或者通知中的 t 值所规定的时间已经过期，则可以判定这个请求或者通知为无效的，这样做可以防止网络重复攻击。t 的格式为 UNIX 时间戳，即从1970年01月01日（UTC/GMT 的午夜）开始所经过的秒数。
- **sign（安全签名）**：sign = MD5(key + t)，即把加密 key 和 t 进行字符串拼接后，计算一下 md5 值。这里的 key 即 CGI 调用 key，您在腾讯云直播管理控制台 域名管理 中可以进行设置，以推流配置为例步骤如下：

1.1 进入 域名管理 单击推流域名后面的 **管理** 。

1.2 选择 **推流配置**，查看 **鉴权配置** 标签，单击 **编辑** 进行配置。



> **说明**
> 若您需要对**播放域名**进行鉴权配置，您可以在 域名管理 选择播放域名或单击后面的 **管理**，进入播放域名详情页，选择**访问控制**，查看 **鉴权配置** 标签，单击 **编辑**，进行配置。

- **安全原理**
  由于 MD5 是不可逆的 HASH 算法，所以只要确保 KEY 不泄露，即使攻击者拿到很多对 t 和 sign 也无法反算出 KEY 值，进而无法进行伪装攻击。
- **计算示例**
  例如，我们现在的时间是2021-07-21 11:46:00，我们希望有效期是1分钟，也就是2021-07-21 11:47:00以后再收到携带这个 t 的请求或者通知即判定为非法的：

```
t = "2021-07-21 11:47:00" = 1626839220
```

假设我们的 key 是 **5d41402abc4b2a76b9719d911017c592**，那么我们计算的签名结果就是：

```
sign = MD5(5d41402abc4b2a76b9719d911017c5921626839220) =
5ee8ca6c28cbe415b40352969cdf8249
```

错误码

HTTP 错误

错误码	含义	备注
403	Forbidden	接口为了安全考虑开启了校验，若使用浏览器验证发现该错误，可检查下 cookie 里是否含有 skey。
404	Not Found	查看请求时是否带上 host。

接口通用返回错误

错误信息	含义
appid is invalid	appid 不合法，表示未开通该功能。

接口前端接入返回错误信息

错误信息	含义
cmd is invalid	cmd 不合法，表示未开通该功能。
sign invalid	鉴权计算错误，请参见 安全机制 。
time expired	鉴权成功，但是超过了 URL 有效期，请参见 安全机制 。

接口后端查询返回错误码

错误码	错误信息	含义
0	query data successfully	本次查询成功，并返回结果数据。
1000	user is not registered for statapi	用户没有注册 statapi，请提工单到后台开通。
1001	user service for statapi was stopped	用户 statapi 访问服务已经被终止。
1201	internel/system error	内部系统错误，属于系统异常，建议通过 工单 反馈到服务商。
1202	invalid request/request frequency exceeds limit	无效的请求，一般是超过了频控次数，如果频率不能满足业务需要，可申请增加次数。
1204	invalid input param	输入参数错误，请检查下输入的参数是否符合接口规范。

1301	has not live stream	没有活跃的流，在调用实时接口时会返回该错误码。
10003	query data is empty	后端查询数据成功，但是返回数据为空。例如，某时间段没有播放，此时调用接口 Get_LivePlayStatHistory 就会返回10003。

注意

以上错误码针对本文 API 列表中的 API，不包括 消息事件通知 。

消息通知

详情请参见腾讯云事件 消息通知 服务。

录制和回看

最近更新时间：2025-06-24 18:08:51

功能介绍

录制回看是指您可以把用户整个直播过程录制下来，然后作为点播视频用于回看。
在 App 上线的初期阶段，由于主播数量比较少，所以在直播列表中加入录制回看，能够在一定程度上丰富 App 在观众端的信息量。即使到 App 成长起来主播数量形成规模以后，好的直播内容的沉淀依然是必不可少的一个部分，每个主播的个人介绍里除了有名字、照片和个人信息，历史直播的视频回看更是不可或缺的一个重要组成部分。

开启录制

录制回看功能依托于腾讯云的**云点播服务**支撑，如果您想要对接这个功能，首先需要在腾讯云的管理控制台 开通云点播服务 。服务开通之后，新录制的文件就可以在云点播控制台的 视频管理 里找到它们。
开启录制的方法如下：

> **说明：**
> - 如需通过 API 对直播频道进行录制，详细请参见 创建录制任务 。
> - 录制转点播后，文件自动存放于点播平台，故用户需在使用录制功能前，提前开通点播服务并购买相关空间和流量用于存放和播放录制后的视频文件，详细请参见 点播快速入门 。

基本步骤

在云直播控制台菜单栏内选择**功能配置** > **直播录制**，单击**创建模板**进行设置。设置完基本信息后，单击**保存** 。具体操作请参见**录制模板配置** 。

配置项	配置描述
单个录制文件时长（分钟）	录制 HLS 格式最长单个文件时长无限制，如果超出续录等待时长则新建文件继续录制。录制 MP4、FLV 或 AAC 格式单个文件时长限制为1分钟 – 120分钟。
续录等待时长（秒）	仅 HLS 格式支持文件推流中断续录，续录等待时长可设置为0s – 1800s。

保存时长（天）		单个录制文件保存最大时长均为1500天，文件保存时长0为永久。可选择永久存储或指定时间。
指定点播应用/分类		支持录制至云点播 VOD 指定 子应用 的点播分类中，默认录制至账号主应用，仅支持写入状态为开启的子应用。
高级配置	视频存储策略	支持对录制至云点播上的媒资进行降冷操作，若录制文件不需要频繁访问，可以使用降冷功能来实现低频访问长期存储。若录制视频为正常业务回放需要，标准存储即可满足需求，默认为标准存储。 • 选择**标准存储**时，若目前选中的应用已开启降冷策略，录制文件会先生成标准存储文件后再根据降冷策略进行降冷，可 查看策略 。 • 选择**低频存储**时，若目前选中的应用/分类已开启降冷策略，录制文件会先直接生成低频存储文件后再判断是否执行点播降冷策略。
	点播任务流处理	单击**选择绑定的任务流** 可选择绑定点播子应用下已建立的任务流，或从当前点播任务流选择界面点击任务流名称前往点播控制台新增/修改任务流配置。绑定成功后，在生成录制文件后执行点播任务流模板，产生对应的 云点播费用 。

关联域名

创建好录制模板后，需在 域名管理 中，选择对应的推流域名，进入**模板配置**栏，单击**录制配置 > 编辑**为该域名指定录制模板后，单击**保存**即可。更多详情请参见 关联录制模板 。

获取文件

一个新的录制视频文件生成后，会相应的生成一个观看地址，您可以按照自己的业务需求对其进行处理。在小直播中，我们直接将录制的文件 URL 和房间列表拼在了一起，以弥补在线主播不足的窘境。
但您可以结合自己的业务场景实现很多的扩展功能，例如：您可以将其追加到主播的资料信息里，作为该主播曾经直播的节目而存在；或者将其放入回放列表中，经过专门的人工筛选，将优质的视频推荐给您的 App 用户。

如何才能拿到文件的地址，有如下两种解决方案：

方案1：被动监听通知

您可以使用腾讯云的 回调配置：您的服务器注册一个自己的回调 URL 给腾讯云，腾讯云会在一个新的录制文件生成时通过这个 URL 通知给您。
在云直播菜单栏内选择**功能配置** > **直播回调**，单击**创建模板**。在回调设置弹框中填写完成回调信息，单击**保存**即可。更多详情请参见 创建回调模板。

关联域名
创建好回调模板后，需在 域名管理 中，选择对应的推流域名，进入**模板配置**栏，单击**回调配置** > **编辑** 为该域名指定回调模板后，单击**确定**即可。

如下是一个典型的通知消息，它的含义是：一个 ID 为 `9192487266581821586` 的新的 FLV 录制文件已经生成，播放地址为：`http://200025724.vod.myqcloud.com/200025724_ac92b781a22c4a3e937c9e61c2624af7.f0.flv` 。

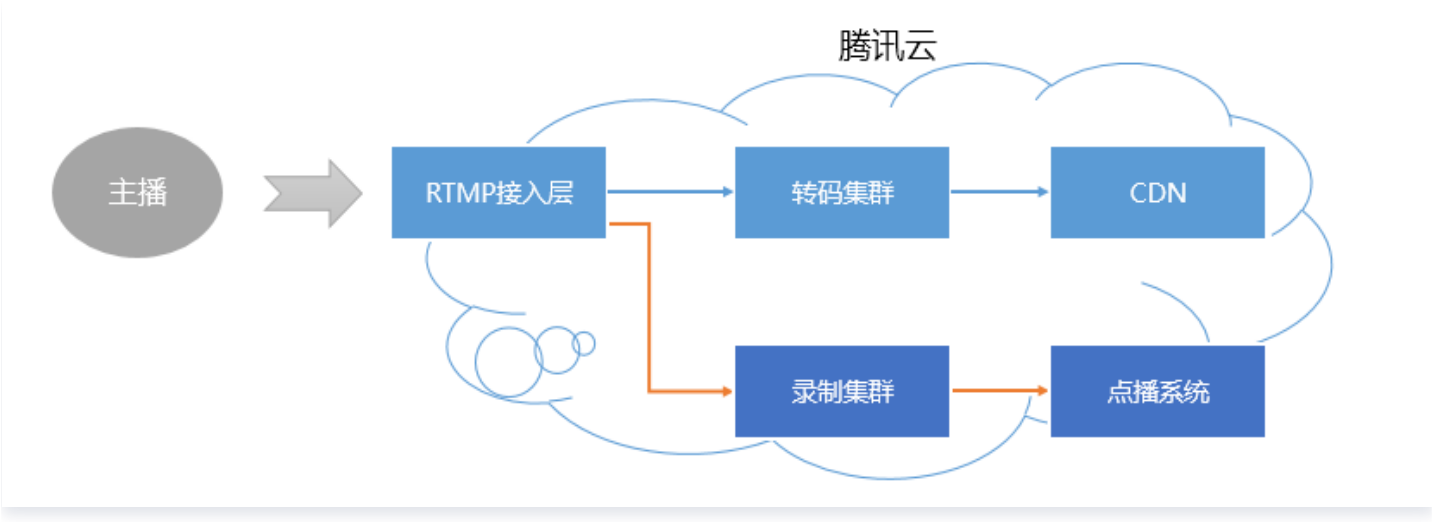
```
{
    "channel_id": "2121_15919131751",
    "end_time": 1473125627,
    "event_type": 100,
    "file_format": "flv",
    "file_id": "9192487266581821586",
    "file_size": 9749353,
    "sign": "fef79a097458ed80b5f5574cbc13e1fd",
    "start_time": 1473135647,
    "stream_id": "2121_15919131751",
    "t": 1473126233,
    "video_id": "200025724_ac92b781a22c4a3e937c9e61c2624af7",
    "video_url":
"http://200025724.vod.myqcloud.com/200025724_ac92b781a22c4a3e937c9e61c2624af7.f0.f
lv"
}
```

方案2：主动查询获取

录制文件生成后自动存储到点播系统，您可以直接在点播系统查看，或直接通过点播 API 查询，详情请参见 录制文件获取 。

常见问题

1. 直播录制的原理是什么?



对于一条直播流，一旦开启录制，音视频数据就会被旁路到录制系统。主播的手机推上来的每一帧数据，都会被录制系统追加写入到录制文件中。
一旦直播流中断，接入层会立刻通知录制服务器将正在写入的文件落地，将其转存到点播系统中，并为其生成索引，这样您在点播系统中就会看到这个新生成的录制文件了。同时，如果您配置了录制事件通知，录制系统会将该文件的**索引 ID** 和**在线播放地址**等信息通知给您之前配置的服务器上。
但是，如果一个文件过大，在云端的转出和处理过程中就很容易出错，所以为了确保成功率，我们的单个录制文件最长不会超过120分钟，您可以通过 RecordInterval 参数指定更短的分片。

2. 一次直播会有几个录制文件?

- **录制 MP4、FLV 或 AAC 格式**：单个文件时长限制为1分钟 – 120分钟。您可以通过 创建录制模板 接口中的 RecordInterval 参数指定更短的分片。
  - 如果一次直播过程非常短暂，录制模块未启动就结束推流，那么系统会无法生成录制文件。
  - 如果一次直播时间不算长（小于 RecordInterval），且中途没有推流中断的事情发生，那么通常只有一个文件。
  - 如果一次直播时间很长（超过 RecordInterval ），那么会按照 RecordInterval 指定的时间长度进行分片，分片的原因是避免过长的文件在分布式系统中流转时间的不确定性。
  - 如果一次直播过程中发生推流中断（之后 SDK 会尝试重新推流），那么每次中断均会产生一个新的分片。
- **录制 HLS 格式**：最长单个文件时长无限制，如果超出续录超时时间则新建文件继续录制。续录超时时长可设置为0s – 1800s。

3. 如何知道哪些文件属于某一次直播?

准确来说，作为 PAAS 的腾讯云并不清楚您的一次直播是怎么定义的，如果您的一次直播持续了20分钟，但中间有一次因为网络切换导致的断流，以及一次手动的停止和重启，那么这算是一次直播还是三次呢?

对于普通的移动直播场景，我们一般定义如下的界面之间的这段时间为一次直播：

所以来自 App 客户端的时间信息很重要，如果您希望定义这段时间内的录制文件都属于这次直播，那么只需要用直播码和时间信息检索收到的录制通知即可（每一条录制通知事件都会携带 StreamID、开始时间和结束时间等信息）。