移动直播 SDK

直播推流

产品文档

腾讯云

文档目录

直播推流
- iOS
  - 摄像头推流
  - 录屏推流
- Android
  - 摄像头推流
  - 录屏推流
- Web 推流

直播推流
iOS
摄像头推流

最近更新时间：2022-06-14 12:54:59

功能概述

摄像头推流，是指采集手机摄像头的画面以及麦克风的声音，进行编码之后再推送到直播云平台上。腾讯云 LiteAVSDK 通过 V2TXLivePusher 接口提供摄像头推流能力。

特别说明

x86 模拟器调试：由于 SDK 大量使用 iOS 系统的音视频接口，这些接口在 Mac 上自带的 x86 仿真模拟器下往往不能工作。所以，如果条件允许，推荐您尽量使用真机调试。

示例代码

所属平台	GitHub 地址	关键类
iOS	Github	CameraPushViewController.m
Android	Github	CameraPushMainActivity.java

说明：
除上述示例外，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。

- iOS：MLVB-API-Example
- Android：MLVB-API-Example

功能对接

1. 下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

2. 给 SDK 配置 License 授权

单击 License 申请 获取测试用的 License，您会获得两个字符串：一个字符串是 licenseURL，另一个字符串是解密 key。

在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (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]);
}
@end
```

3. 初始化 V2TXLivePusher 组件

创建一个 `V2TXLivePusher` 对象，需要指定对应的 `V2TXLiveMode` 。

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP]; // 指定对应的直播协议为 RTMP
```

4. 开启摄像头预览

想要开启摄像头预览，首先需要调用 V2TXLivePusher 中的 `setRenderView` 接口，改接口需要设置一个用于显示视频画面的 view 对象，然后调用 `startCamera` 接口开启当前手机摄像头的预览。

```
//创建一个 view 对象，并将其嵌入到当前界面中
UIView *_localView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view insertSubview:_localView atIndex:0];
_localView.center = self.view.center;
//启动本地摄像头预览
[_pusher setRenderView:_localView];
[_pusher startCamera:YES];
```

> 注意：
> 如果要给 view 增加动画效果，需要修改 view 的 transform 属性而不是 frame 属性。

```
[UIView animateWithDuration:0.5 animations:^{
_localView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
}];
```

5. 启动和结束推流

如果已经通过 `startCamera` 接口启动了摄像头预览，就可以调用 V2TXLivePusher 中的 startPush 接口开始推流。推流地址可以使用 TRTC 地址，或者使用 RTMP 地址，前者使用 UDP 协议，推流质量更高，并支持连麦互动。

```
//启动推流， URL 可以使用 trtc:// 或者 rtmp:// 两种协议，前者支持连麦功能
NSString* url = @"trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&user
Id=A&usersig=xxxxx"; //支持连麦
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
//不支持连麦，直接推流到直播 CDN
[_pusher startPush:url];
```

推流结束后，可以调用 V2TXLivePusher 中的 stopPush 接口结束推流。

```
//结束推流
[_pusher stopPush];
```

> 注意：
> 如果已经启动了摄像头预览，请在结束推流时将其关闭。

- **如何获取可用的推流 URL？**
  开通直播服务后，可以使用【直播控制台】>【辅助工具】>【地址生成器】生成推流地址，详细信息请参见 推拉流 URL。

- **返回 V2TXLIVE_ERROR_INVALID_LICENSE 的原因**
  如果 `startPush` 接口返回 `V2TXLIVE_ERROR_INVALID_LICENSE`，则代表您的 License 校验失败了，请检查 第2步：给 SDK 配置 License 授权 中的工作是否有问题。

6. 纯音频推流

如果您的直播场景是纯音频直播，不需要视频画面，那么您可以不执行 第4步 中的操作，或者在调用 `startPush` 之前不调用 `startCamera` 接口即可。

```
V2TXLivePusher *_pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
//启动推流， URL 可以使用 trtc:// 或者 rtmp:// 两种协议，前者支持连麦功能
NSString* url = @"trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx"; //支持连麦
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
//不支持连麦，直接推流到直播 CDN
[_pusher startPush:url];
[_pusher startMicrophone];
```

> 说明：
> 如果您启动纯音频推流，但是 RTMP、FLV 、HLS 格式的播放地址拉不到流，那是因为线路配置问题，请 提工单 联系我们帮忙修改配置。

7. 设定画面清晰度

调用 V2TXLivePusher 中的 setVideoQuality 接口，可以设定观众端的画面清晰度。之所以说是观众端的画面清晰度，是因为主播看到的视频画面是未经编码压缩过的高清原画，不受设置的影响。而 `setVideoQuality` 设定的视频编码器的编码质量，观众端可以感受到画质的差异。详情请参见 设定画面质量。

8. 美颜美白和红润特效

调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美颜效果。

美颜风格

SDK 内置三种不同的磨皮算法，每种磨皮算法即对应一种美颜风格，您可以选择最适合您产品定位的方案。详情请参见 TXBeautyManager.h 文件：

美颜风格	效果说明
TXBeautyStyleSmooth	光滑，适用于美女秀场，效果比较明显
TXBeautyStyleNature	自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然
TXBeautyStylePitu	由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高

美颜风格可以通过 TXBeautyManager 的 setBeautyStyle 接口设置：

美颜风格	设置方式	接口说明
美颜级别	通过 TXBeautyManager 的 `setBeautyLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显
美白级别	通过 TXBeautyManager 的 `setWhitenessLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显
红润级别	通过 TXBeautyManager 的 `setRuddyLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显

9. 色彩滤镜效果

- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
- 调用 TXBeautyManager 的 `setFilterStrength` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

从手机 QQ 和 Now 直播的经验来看，单纯通过 TXBeautyManager 的 `setBeautyStyle` 调整美颜风格是不够的，只有将美颜风格和 `setFilter` 配合使用才能达到更加丰富的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜供您使用。

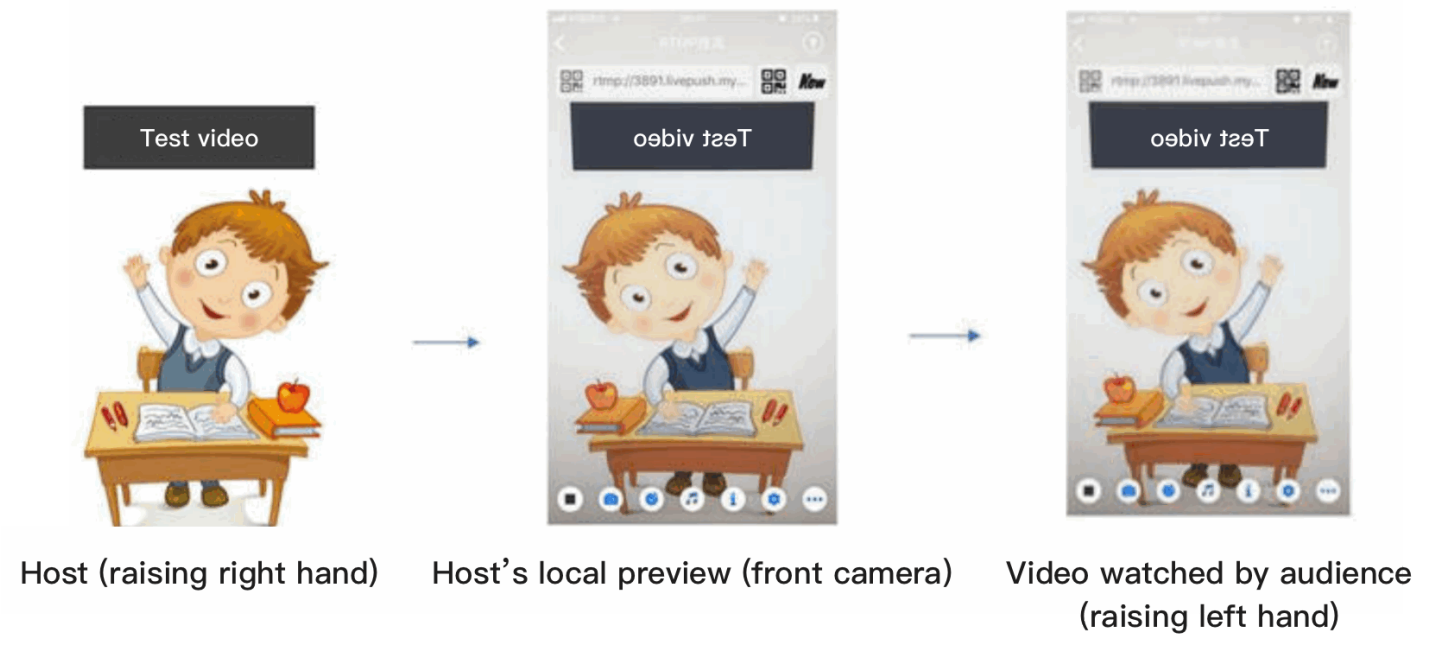
```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofTyp
e:@"bundle"];
path = [path stringByAppendingPathComponent:lookupFileName];
UIImage *image = [UIImage imageWithContentsOfFile:path];
[[_pusher getBeautyManager] setFilter:image];
[[_pusher getBeautyManager] setFilterStrength:0.5f];
```

10. 设备管理

V2TXLivePusher 提供了一组 API 用户控制设备的行为，您可通过 `getDeviceManager` 获取 TXDeviceManager 实例进一步进行设备管理，详细用法请参见 TXDeviceManager API。

11. 观众端的镜像效果

通过调用 V2TXLivePusher 的 setRenderMirror 可以改变摄像头的镜像方式，继而影响观众端观看到的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，默认情况下自己看到的画面会被 SDK 反转。



Host (raising right hand) | Host's local preview (front camera) | Video watched by audience (raising left hand)

12. 横屏推流

大多数情况下，主播习惯以“竖屏持握”手机进行直播拍摄，观众端看到的也是竖屏分辨率的画面（例如 540 × 960 这样的分辨率）；有时主播也会“横屏持握”手机，这时观众端期望能看到是横屏分辨率的画面（例如 960 × 540 这样的分辨率）。

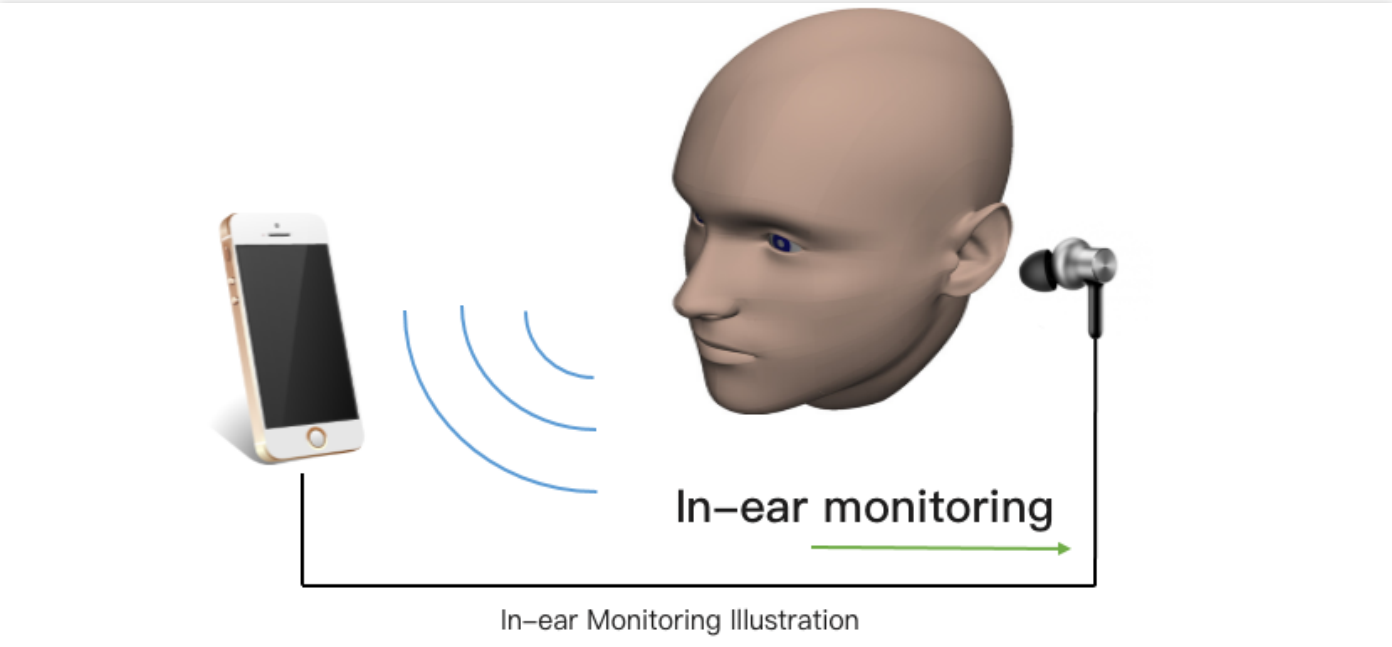
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
[_pusher setVideoQuality:videoQuality
resolutionMode:isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideo
ResolutionModePortrait];
```

13. 音效设置

调用 V2TXLivePusher 中的 getAudioEffectManager 获取 TXAudioEffectManager 实例可以实现背景混音、耳返、混响等音效功能。背景混音是指主播在直播时可以选取一首歌曲伴唱，歌曲会在主播的手机端播放出来，同时也会被混合到音视频流中被观众端听到，所以被称为“混音”。

- 调用 TXAudioEffectManager 中的 `enableVoiceEarMonitor` 选项可以开启耳返功能，“耳返”指的是当主播带上耳机来唱歌时，耳机中要能实时反馈主播的声音。
- 调用 TXAudioEffectManager 中的 `setVoiceReverbType` 接口可以设置混响效果，例如 KTV、会堂、磁性、金属等，这些效果也会作用到观众端。
- 调用 TXAudioEffectManager 中的 `setVoiceChangerType` 接口可以设置变调效果，例如“萝莉音”，“大叔音”等，用来增加直播和观众互动的趣味性，这些效果也会作用到观众端。



In-ear Monitoring Illustration

说明：
详细用法请参见 TXAudioEffectManager API。

14. 设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数 `(x, y, scale)` 所决定。

- SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。
- `(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`(0.1, 0.1, 0.1)`，那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

```
//设置视频水印
[_pusher setWatermark:[UIImage imageNamed:@"watermark"] x:0.03 y:0.015 scale:1];
```

15. 主播端弱网提醒

手机连接 Wi-Fi 网络不一定就非常好，如果 Wi-Fi 信号差或者出口带宽很有限，可能网速不如4G，如果主播在推流时遇到网络很差的情况，需要有一个友好的提示，提示主播应当切换网络。

通过 V2TXLivePusherObserver 里的 onWarning 可以捕获 **V2TXLIVE_WARNING_NETWORK_BUSY** 事件，它代表当前主播的网络已经非常糟糕，出现此事件即代表观众端会出现卡顿。此时就可以像上图一样在 UI 上弹出一个“弱网提示”。

```
- (void)onWarning:(V2TXLiveCode)code
message:(NSString *)msg
extraInfo:(NSDictionary *)extraInfo {
dispatch_async(dispatch_get_main_queue(), ^{
if (code == V2TXLIVE_WARNING_NETWORK_BUSY) {
[_notification displayNotificationWithMessage:
@"您当前的网络环境不佳，请尽快更换网络保证正常直播" forDuration:5];
}
});
}
```

16. 发送 SEI 消息

调用 V2TXLivePusher 中的 sendSeiMessage 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

- 答题直播：推流端将题目下发到观众端，可以做到“音-画-题”完美同步。
- 秀场直播：推流端将歌词下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
- 在线教育：推流端将激光笔和涂鸦操作下发到观众端，可以在播放端实时地划圈划线。

由于自定义消息是直接被塞入视频数据中的，所以不能太大（几个字节比较合适），一般常用于塞入自定义的时间戳等信息。

```
int payloadType = 5;
NSString* msg = @"test";
[_pusher sendSeiMessage:payloadType data:[msg dataUsingEncoding:NSUTF8StringEncod
ing]];
```

常规开源播放器或者网页播放器是不能解析 SEI 消息的，必须使用 LiteAVSDK 中自带的 V2TXLivePlayer 才能解析这些消息：

1. 设置：

```
int payloadType = 5;
[_player enableReceiveSeiMessage:YES payloadType:payloadType];
```

2. 当 V2TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 V2TXLivePlayerObserver 中的 onReceiveSeiMessage 回调来接收该消息。

事件处理

事件监听

SDK 通过 V2TXLivePusherObserver 代理来监听推流相关的事件通知和错误通知，详细的事件表和错误码表请参见 错误码表 。

错误通知

SDK 发现部分严重问题，推流无法继续。

事件 ID	数值	含义说明
V2TXLIVE_ERROR_FAILED	-1	暂未归类的通用错误
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	调用 API 时，传入的参数不合法
V2TXLIVE_ERROR_REFUSED	-3	API 调用被拒绝
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	当前 API 不支持调用
V2TXLIVE_ERROR_INVALID_LICENSE	-5	license 不合法，调用失败
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	请求服务器超时
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	服务器无法处理您的请求

警告事件

SDK 发现部分警告问题，但 WARNING 级别的事件都会触发一些尝试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复。

事件 ID	数值	含义说明
V2TXLIVE_WARNING_NETWORK_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	视频回放期间出现滞后
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	摄像头打开失败
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	摄像头正在被占用中，可尝试打开其他摄像头
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	麦克风打开失败
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了

事件 ID	数值	含义说明
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	当前系统不支持屏幕分享
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断

录屏推流

最近更新时间：2022-06-14 12:56:05

概述

录屏功能是 iOS 10 新推出的特性，苹果在 iOS 9 的 ReplayKit 保存录屏视频的基础上，增加了视频流实时直播功能，官方介绍见 Go Live with ReplayKit。iOS 11 增强为 ReplayKit2，进一步提升了 Replaykit 的易用性和通用性，并且可以对整个手机实现屏幕录制，并非只是支持 ReplayKit 功能，因此录屏推流建议直接使用 iOS 11 的 ReplayKit2 屏幕录制方式。系统录屏采用的是扩展方式，扩展程序有单独的进程，iOS 系统为了保证系统流畅，给扩展程序的资源相对较少，扩展程序内存占用过大也会被 Kill 掉。腾讯云 LiteAV SDK 在原有直播的高质量、低延迟的基础上，进一步降低系统消耗，保证了扩展程序稳定。

> 注意：
> 本文主要介绍 iOS 11 的 ReplayKit2 录屏使用 SDK 推流的方法，涉及 SDK 的使用介绍同样适用于其它方式的自定义推流。更详细的使用说明可以参考 Demo 里 TXReplayKit_Screen 文件夹示例代码。

功能体验

体验 iOS 录屏可以扫码进行安装视频云工具包。

> 注意：
> 录屏推流功能仅11.0以上系统可体验。

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

使用步骤

1. 打开控制中心，长按屏幕录制按钮，选择 **视频云工具包**。
2. 打开 **视频云工具包** > **推流演示（录屏推流）**，输入推流地址或单击 **New** 自动获取推流地址，单击 **开始推流**。

推流设置成功后，顶部通知栏会提示推流开始，此时您可以在其它设备上看到该手机的屏幕画面。单击手机状态栏的红条，即可停止推流。

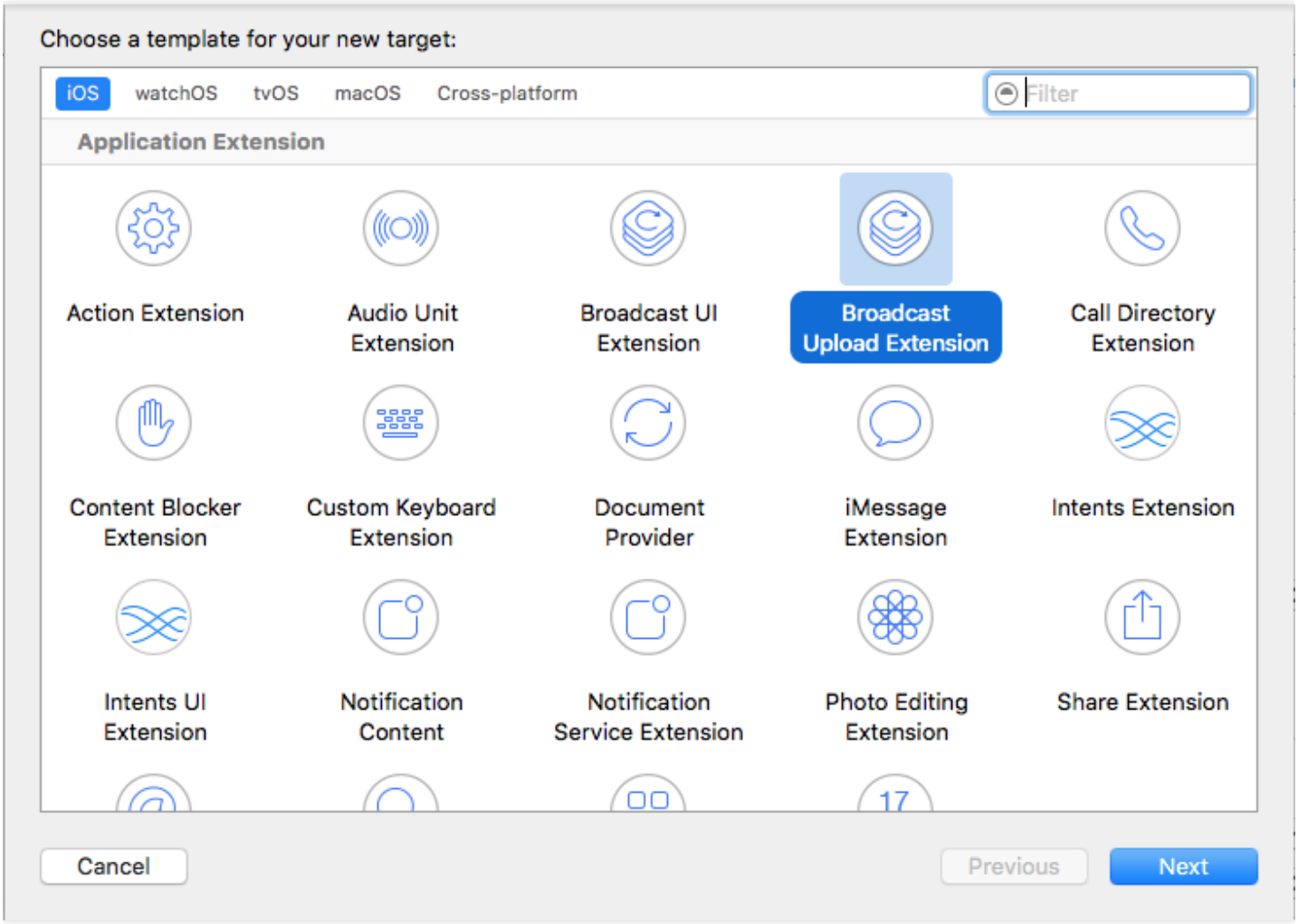
开发环境准备

Xcode 准备

Xcode 9 及以上的版本，手机也必须升级至 iOS 11 以上，否则模拟器无法使用录屏特性。

创建直播扩展

在现有工程选择 **New** > **Target...**，选择 **Broadcast Upload Extension**，如图所示。



配置好 Product Name。单击 **Finish** 后可以看到，工程多了所输 Product Name 的目录，目录下有个系统自动生成的 SampleHandler 类，这个类负责录屏的相关处理。

导入 LiteAV SDK

直播扩展需要导入 TXLiteAVSDK.framework。扩展导入 framework 的方式和主 App 导入方式相同，SDK 的系统依赖库也没有区别。具体请参见腾讯云官网 工程配置（iOS）。

对接流程

步骤1：创建推流对象

在 SampleHandler.m 中添加下面代码：

```
#import "SampleHandler.h"
#import "V2TXLivePusher.h"

static V2TXLivePusher *s_txLivePublisher;
static NSString *s_rtmpUrl;
```

```
- (void)initPublisher {
if (s_txLivePublisher) {
[s_txLivePublisher stopPush];
}
s_txLivePublisher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
[s_txLivePublisher setObserver:self];
[s_txLivePublisher startPush:s_rtmpUrl];
}
```

- s_txLivePublisher 是我们用于推流的对象，因为系统录屏回调的 sampleHandler 实例有可能不只一个，因此对变量采用静态声明，确保录屏推流过程中使用的是同一个推流器。
- 实例化 s_txLivePublisher 的最佳位置是在 `-[SampleHandler broadcastStartedWithSetupInfo:]` 方法中，直播扩展启动后会回调这个函数，就可以进行推流器初始化开始推流。但在 ReplayKit2 的屏幕录制扩展启动时，回调给 s_txLivePublisher 的 setupInfo 为 nil，无法获取启动推流所需要的推流地址等信息，因此通常回调此函数时发通知给主 App，在主 App 中设置好推流地址，横竖屏清晰度等信息后再传递给扩展并通知扩展启动推流。
- 扩展与主 App 间的通信请参见 扩展与宿主 App 之间的通信与数据传递方式 。

步骤2：横屏推流与分辨率设置

手机录屏直播提供了多个级别的分辨率可供选择。setVideoQuality 方法用来设置分辨率及横竖屏推流，以下录屏推流分辨率与横屏推流设置示例：

```
static BOOL s_landScape; //YES:横屏,  NO:竖屏
[s_txLivePublisher setVideoQuality:V2TXLiveVideoResolution960x540
resolutionMode:s_landScape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideo
ResolutionModePortrait];
[s_txLivePublisher startPush:s_rtmpUrl];
```

步骤3：发送视频

Replaykit 会将视频以回调的方式传给 `-[SampleHandler processSampleBuffer:withType]` 。

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBuf
ferType)sampleBufferType {
switch (sampleBufferType) {
```

```
case RPSampleBufferTypeVideo:
// Handle video sample buffer
{
if (!CMSampleBufferIsValid(sampleBuffer))
return;
//保存一帧在 startPush 时发送,防止推流启动后或切换横竖屏因无画面数据而推流不成功
if (s_lastSampleBuffer) {
CFRelease(s_lastSampleBuffer);
s_lastSampleBuffer = NULL;
}
s_lastSampleBuffer = sampleBuffer;
CFRetain(s_lastSampleBuffer);
V2TXLiveVideoFrame *videoFrame = [V2TXLiveVideoFrame new];
videoFrame.bufferType = V2TXLiveBufferTypePixelBuffer;
videoFrame.pixelFormat = V2TXLivePixelFormatNV12;
videoFrame.pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
videoFrame.rotation = V2TXLiveRotation0;
[s_txLivePublisher sendCustomVideoFrame:videoFrame];
}
}
```

视频 sampleBuffer 只需要调用 `-[V2TXLivePusher sendCustomVideoFrame:]` 发送即可。

系统分发视频 sampleBuffer 的频率并不固定，如果画面静止，可能很长时间才会有一帧数据过来。SDK 考虑到这种情况，内部会做补帧逻辑。

> 注意：
> 建议保存一帧给推流启动时使用，防止推流启动或切换横竖屏时因无新的画面数据采集发送，因为画面没有变化时系统可能会很长时间才采集一帧画面。

步骤4：设置 Logo 水印

据相关政策规定，直播视频必须加上水印。腾讯视频云目前支持两种水印设置方式：一种是在推流 SDK 进行设置，原理是在 SDK 内部进行视频编码前就给画面打上水印。另一种方式是在云端打水印，也就是云端对视频进行解析并添加水印 Logo。

这里我们特别建议您使用 SDK 添加水印，因为在云端打水印有三个明显的问题：

- 这是一种很耗云端机器的服务，而且不是免费的，会拉高您的费用成本。
- 在云端打水印对于推流期间切换分辨率等情况的兼容并不理想，会有很多花屏的问题发生。
- 在云端打水印会引入额外的3s以上的视频延迟，这是转码服务所引入的。

SDK 所要求的水印图片格式为 PNG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（建议您不要在 Windows 下将 JPG 格式的图片修改后缀名就直接使用，因为专业的 PNG 图标都是需要由专业的美工设计师处理的）。

```
//设置视频水印
[s_txLivePublisher setWatermark:image x:0 y:0 scale:1];
```

步骤5：结束推流

结束推流 ReplayKit 会调用 `-[SampleHandler broadcastFinished]`，示例代码：

```
- (void)broadcastFinished {
// User has requested to finish the broadcast.
if (s_txLivePublisher) {
[s_txLivePublisher stopPush];
s_txLivePublisher = nil;
}
}
```

因为用于推流的 `V2TXLivePusher` 对象同一时刻只能有一个在运行，所以结束推流时要做好清理工作。

事件处理

1. 事件监听

SDK 通过 V2TXLivePusherObserver 代理来监听推流相关的事件通知和错误通知，详细的事件表和错误码表请参见错误码表。

2. 错误通知

SDK 发现部分严重问题，推流无法继续

事件 ID	数值	含义说明
V2TXLIVE_ERROR_FAILED	-1	暂未归类的通用错误。
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	调用 API 时，传入的参数不合法。
V2TXLIVE_ERROR_REFUSED	-3	API 调用被拒绝。
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	当前 API 不支持调用。
V2TXLIVE_ERROR_INVALID_LICENSE	-5	license 不合法，调用失败。

事件 ID	数值	含义说明
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	请求服务器超时。
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	服务器无法处理您的请求。

3. 警告事件

SDK 发现部分警告问题，但 WARNING 级别的事件都会触发一些尝试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复。

事件 ID	数值	含义说明
V2TXLIVE_WARNING_NETWORK_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻。
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	当前视频播放出现卡顿
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	摄像头打开失败。
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	摄像头正在被占用中，可尝试打开其他摄像头。
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	麦克风打开失败。
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败。
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	当前系统不支持屏幕分享。
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了。
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断。

附: 扩展与宿主 App 之间的通信与数据传递方式参考

ReplayKit2 录屏只唤起 upload 直播扩展，直播扩展不能进行 UI 操作，也不适于做复杂的业务逻辑，因此通常宿主 App 负责鉴权及其它业务逻辑，直播扩展只负责进行屏幕的音画采集与推流发送，扩展就经常需要与宿主 App 之间进行数据传递与通信。

1. 发本地通知

扩展的状态需要反馈给用户，有时宿主 App 并未启动，此时可通过发送本地通知的方式进行状态反馈给用户与激活宿主 App 进行逻辑交互，如在直播扩展启动时通知宿主 App：

```
- (void)broadcastStartedWithSetupInfo:(NSDictionary<nsstring *,nsobject="" *="">
*)setupInfo {
[self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"录屏已开始，请
从这里单击回到Demo->录屏幕推流->设置推流URL与横竖屏和清晰度" userInfo:@{kReplayKit2Uploa
dingKey: kReplayKit2Uploading}];
}

- (void)sendLocalNotificationToHostAppWithTitle:(NSString*)title msg:(NSString*)
msg userInfo:(NSDictionary*)userInfo
{
UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotification
Center];

UNMutableNotificationContent* content = [[UNMutableNotificationContent alloc] in
it];
content.title = [NSString localizedUserNotificationStringForKey:title arguments:
nil];
content.body = [NSString localizedUserNotificationStringForKey:msg arguments:nil
];
content.sound = [UNNotificationSound defaultSound];
content.userInfo = userInfo;

// 在设定时间后推送本地推送
UNTimeIntervalNotificationTrigger* trigger = [UNTimeIntervalNotificationTrigger
triggerWithTimeInterval:0.1f repeats:NO];

UNNotificationRequest* request = [UNNotificationRequest requestWithIdentifier:
@"ReplayKit2Demo"
content:content trigger:trigger];

//添加推送成功后的处理！
[center addNotificationRequest:request withCompletionHandler:^(NSError * _Nullab
le error) {

}];
}
```

通过此通知可以提示用户回到主 App 设置推流信息、启动推流等。

2. 进程间的通知 CFNotificationCenter

扩展与宿主 App 之间还经常需要实时的交互处理，本地通知需要用户点击横幅才能触发代码处理，因此不能通过本地通知的方式。而 NSNotificationCenter 不能跨进程，因此可以利用 CFNotificationCenter 在宿主 App 与扩展之前通知发送，但此通知不能通过其中的 userInfo 字段进行数据传递，需要通过配置 App Group 方式使用 NSUserDefault 进行数据传递（也可以使用剪贴板，但剪贴板有时不能实时在进程间获取数据，需要加些延迟规避），如主 App 在获取好推流 URL 等后，通知扩展可以进行推流时，可通过 CFNotificationCenter 进行通知发送直播扩展开始推流：

```
CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter
(),
kDarvinNotificationNamePushStart,
NULL,
nil,
YES);
```

扩展中可通过监听此开始推流通知，由于此通知是在 CF 层，需要通过 NSNotificationCenter 发送到 Cocoa 类层方便处理：

```
CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
(__bridge const void *)(self),
onDarwinReplayKit2PushStart,
kDarvinNotificationNamePushStart,
NULL,
CFNotificationSuspensionBehaviorDeliverImmediately);

[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handle
ReplayKit2PushStartNotification:) name:@"Cocoa_ReplayKit2_Push_Start" object:nil
];
static void onDarwinReplayKit2PushStart(CFNotificationCenterRef center,
void *observer, CFStringRef name,
const void *object, CFDictionaryRef
userInfo)
{
//转到 cocoa 层框架处理
[[NSNotificationCenter defaultCenter] postNotificationName:@"Cocoa_ReplayKit2_Pu
sh_Start" object:nil];
}
- (void)handleReplayKit2PushStartNotification:(NSNotification*)noti
{
//通过 NSUserDefault 或剪贴板拿到宿主要传递的数据
// NSUserDefaults *defaults = [[NSUserDefaults alloc] initWithSuiteName:kReplayK
it2AppGroupId];
UIPasteboard* pb = [UIPasteboard generalPasteboard];
NSDictionary* defaults = [self jsonData2Dictionary:pb.string];
```

```
s_rtmpUrl = [defaults objectForKey:kReplayKit2PushUrlKey];
s_resolution = [defaults objectForKey:kReplayKit2ResolutionKey];
if (s_resolution.length < 1) {
s_resolution = kResolutionHD;
}
NSString* rotate = [defaults objectForKey:kReplayKit2RotateKey];
if ([rotate isEqualToString:kReplayKit2Portrait]) {
s_landScape = NO;
}
else {
s_landScape = YES;
}
[self start];
}
```

常见问题

ReplayKit2 屏幕录制在 iOS 11 新推出功能，相关的官方文档比较少，且存在着一些问题，使得每个版本的系统都在不断修复完善中。以下是一些使用中的常见现象或问题：

1. **屏幕录制何时自动会停止？**
   系统在锁屏或有电话打入时，会自动停止屏幕录制，此时 SampleHandler 里的 broadcastFinished 函数会被调用，可在此函数发通知提示用户。

2. **采集推流过程中有时屏幕录制会自动停止问题？**
   通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2 的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此 ReplayKit2 上建议推流分辨率不高于720P。

3. **iPhoneX 手机的兼容性与画面变形问题？**
   iPhoneX 手机因为有刘海，屏幕采集的画面分辨率不是 9:16。如果设了推流输出分辨率为 9:16 的比例，如高清里是为 960 × 540 的分辨率，这时因为源分辨率不是 9:16 的，推出去的画面就会稍有变形。建议设置分辨率时根据屏幕分辨率比例来设置，拉流端用 AspectFit 显示模式 iPhoneX 的屏幕采集推流会有黑边是正常现象，AspectFill 看画面会不全。

Android
摄像头推流

最近更新时间：2022-06-14 12:56:40

功能概述

摄像头推流，是指采集手机摄像头的画面以及麦克风的声音，进行编码之后再推送到直播云平台上。腾讯云 LiteAVSDK 通过 V2TXLivePusher 接口提供摄像头推流能力。

特别说明

真机调试： 由于 SDK 大量使用 Android 系统的音视频接口，这些接口在仿真模拟器下往往不能工作，推荐您尽量使用真机调试。

示例代码

所属平台	GitHub 地址	关键类
iOS	Github	CameraPushViewController.m
Android	Github	CameraPushMainActivity.java

说明：
除上述示例外，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。

- iOS：MLVB-API-Example
- Android：MLVB-API-Example

功能对接

1. 下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

2. 给 SDK 配置 License 授权

单击 License 申请 获取测试用 License，您会获得两个字符串：其中一个字符串是 licenceURL，另一个字符串是解密 licenceKey。

在您的 App 调用企业版 SDK 相关功能之前（建议在 Application类中）进行如下设置：

```
public class MApplication extends Application {
@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
}
}
```

3. 初始化 V2TXLivePusher 组件

创建一个 `V2TXLivePusher` 对象，该对象负责完成推流的主要工作。

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMod
e.TXLiveMode_RTMP); //指定对应的直播协议为 RTMP
```

4. 开启摄像头预览

想要开启摄像头的预览画面，您需要先给 SDK 提供一个用于显示视频画面的 `TXCloudVideoView` 对象，由于 `TXCloudVideoView` 是继承自 Android 中的 `FrameLayout`，所以您可以：

1. 直接在 xml 文件中添加一个视频渲染控件：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/pusher_tx_cloud_view"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. 通过调用 V2TXLivePusher 中的 `startCamera` 接口开启当前手机摄像头的预览画面。

```
//启动本地摄像头预览
TXCloudVideoView mPusherView = (TXCloudVideoView) findViewById(R.id.pusher_tx_c
loud_view);
```

```
mLivePusher.setRenderView(mPusherView);
mLivePusher.startCamera(true);
```

5. 启动和结束推流

如果已经通过 `startCamera` 接口启动了摄像头预览，就可以调用 V2TXLivePusher 中的 startPush 接口开始推流。推流地址可以使用 TRTC 地址，或者使用 RTMP 地址，前者使用 UDP 协议，推流质量更高，并支持连麦互动。

```
//启动推流， URL 可以使用 trtc:// 或者 rtmp:// 两种协议，前者支持连麦功能
String rtmpURL = "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&user
Id=A&usersig=xxxxx"; //支持连麦
String rtmpURL = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
//不支持连麦，直接推流到直播 CDN
int ret = mLivePusher.startPush(rtmpURL.trim());
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

推流结束后，可以调用 V2TXLivePusher 中的 stopPush 接口结束推流。

```
//结束推流
mLivePusher.stopPush();
```

> 注意：
> 如果已经启动了摄像头预览，请在结束推流时将其关闭。

- **如何获取可用的推流 URL？**
  开通直播服务后，可以使用 【直播控制台】>【直播工具箱】>【地址生成器】 生成推流地址，详细信息请参见 推拉流 URL。
- **返回 V2TXLIVE_ERROR_INVALID_LICENSE 的原因？**
  如果 `startPush` 接口返回 `V2TXLIVE_ERROR_INVALID_LICENSE`，则代表您的 License 校验失败了，请检查 第2步：给 SDK 配置 License 授权 中的工作是否有问题。

6. 纯音频推流

如果您的直播场景是纯音频直播，不需要视频画面，那么您可以不执行 第4步 中的操作，或者在调用 `startPush` 之前调用 `stopCamera` 接口即可。

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP); //指定对应的直播协议为 RTMP
mLivePusher.startMicrophone();
//启动推流， URL 可以使用 trtc:// 或者 rtmp:// 两种协议，前者支持连麦功能
String rtmpURL = "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx"; //支持连麦
String rtmpURL = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
//不支持连麦，直接推流到直播 CDN
int ret = mLivePusher.startPush(rtmpURL.trim());
```

说明：
如果您启动纯音频推流，但是 RTMP、FLV 、HLS 格式的播放地址拉不到流，那是因为线路配置问题，请 提工单 联系我们帮忙修改配置。

7. 设定画面清晰度

调用 V2TXLivePusher 中的 setVideoQuality 接口，可以设定观众端的画面清晰度。之所以说是观众端的画面清晰度，是因为主播看到的视频画面是未经编码压缩过的高清原画，不受设置的影响。而 `setVideoQuality` 设定的视频编码器的编码质量，观众端可以感受到画质的差异。详情请参见 设定画面质量。

8. 美颜美白和红润特效

调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美颜效果。

美颜风格

SDK 内置三种不同的磨皮算法，每种磨皮算法即对应一种美颜风格，您可以选择最适合您产品定位的方案。定义见 `TXLiveConstants.java` 文件：

美颜风格	效果说明
BEAUTY_STYLE_SMOOTH	光滑，适用于美女秀场，效果比较明显
BEAUTY_STYLE_NATURE	自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然
BEAUTY_STYLE_PITU	由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高

美颜风格可以通过 TXBeautyManager 的 `setBeautyStyle` 接口设置：

美颜风格	设置方式	接口说明

美颜级别	通过 TXBeautyManager 的 `setBeautyLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显
美白级别	通过 TXBeautyManager 的 `setWhitenessLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显
红润级别	通过 TXBeautyManager 的 `setRuddyLevel` 设置	取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显

9. 色彩滤镜效果

- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
- 调用 TXBeautyManager 的 `setFilterStrength` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

从手机 QQ 和 Now 直播的经验来看，单纯通过 TXBeautyManager 的 `setBeautyStyle` 调整美颜风格是不够的，只有将美颜风格和 `setFilter` 配合使用才能达到更加丰富的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜供您使用。

```
//选择期望的色彩滤镜文件
Bitmap filterBmp = decodeResource(getResources(), R.drawable.filter_biaozhun);
mLivePusher.getBeautyManager().setFilter(filterBmp);
mLivePusher.getBeautyManager().setFilterStrength(0.5f);
```

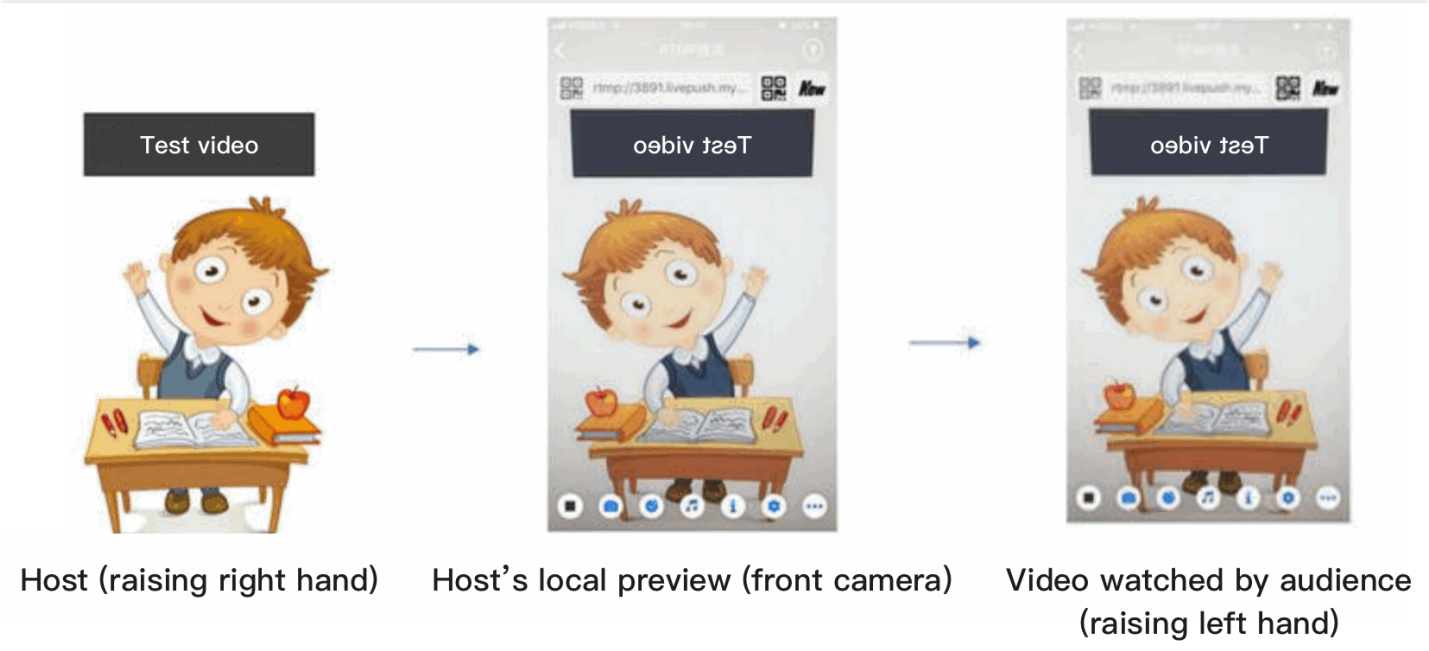
10. 设备管理

V2TXLivePusher 提供了一组 API 用户控制设备的行为。您通过 `getDeviceManager` 获取 TXDeviceManager 实例进一步进行设备管理，详细用法请参见 TXDeviceManager API。

11. 观众端的镜像效果

通过调用 V2TXLivePusher 的 setRenderMirror 可以改变摄像头的镜像方式，继而影响观众端观看到的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，默认情况下自己看到的画面会被 SDK 反转，

这时主播就像照镜子一样，观众看到的效果和主播看到的是一致的。如下图所示：



12. 横屏推流

大多数情况下，主播习惯以“竖屏持握”手机进行直播拍摄，观众端看到的也是竖屏分辨率的画面（例如 540 × 960 这样的分辨率）；有时主播也会“横屏持握”手机，这时观众端期望能看到是横屏分辨率的画面（例如 960 × 540 这样的分辨率）。

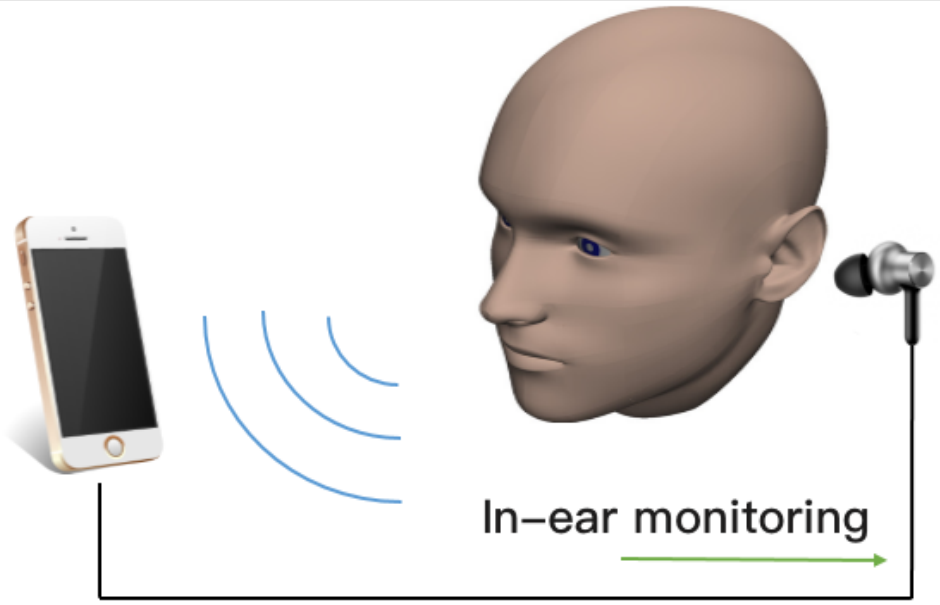
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 `setVideoQuality` 接口的参数来设定观众端的画面横竖屏模式。

```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResoluti
onModeLandscape : V2TXLiveVideoResolutionModePortrait);
```

13. 音效设置

调用 V2TXLivePusher 中的 `getAudioEffectManager` 获取 TXAudioEffectManager 实例可以实现背景混音、耳返、混响等音效功能。背景混音是指主播在直播时可以选取一首歌曲伴唱，歌曲会在主播的手机端播放出来，同时也会被混合到音视频流中被观众端听到，所以被称为“混音”。

- 调用 TXAudioEffectManager 中的 `enableVoiceEarMonitor` 选项可以开启耳返功能，“耳返”指的是当主播带上耳机来唱歌时，耳机中要能实时反馈主播的声音。
- 调用 TXAudioEffectManager 中的 `setVoiceReverbType` 接口可以设置混响效果，例如 KTV、会堂、磁性、金属等，这些效果也会作用到观众端。
- 调用 TXAudioEffectManager 中的 `setVoiceChangerType` 接口可以设置变调效果，例如“萝莉音”，“大叔音”等，用来增加直播和观众互动的趣味性，这些效果也会作用到观众端。



In-ear Monitoring Illustration

说明：
详细用法请参见 TXAudioEffectManager API。

14. 设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数 `(x, y, scale)` 所决定。

- SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。
- `(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`(0.1, 0.1, 0.1)`，则水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

```
//设置视频水印
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 0.03f, 0.015f, 1f);
```

15. 主播端弱网提醒

如果主播在推流时遇到网络很差的情况，需要有一个友好的提示，提示主播应当检查网络。
通过 V2TXLivePusherObserver 里的 onWarning 可以捕获 **V2TXLIVE_WARNING_NETWORK_BUSY** 事件，它代

表当前主播的网络已经非常糟糕，出现此事件即代表观众端会出现卡顿。此时可以在 UI 上弹出一个“弱网提示”来强提醒主播检查网络。

```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
showNetBusyTips(); // 显示网络繁忙的提示
}
}
```

16. 发送 SEI 消息

调用 V2TXLivePusher 中的 sendSeiMessage 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

- 答题直播：推流端将题目下发到观众端，可以做到“音-画-题”完美同步。
- 秀场直播：推流端将歌词下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
- 在线教育：推流端将激光笔和涂鸦操作下发到观众端，可以在播放端实时地划圈划线。

由于自定义消息是直接被塞入视频数据中的，所以不能太大（几个字节比较合适），一般常用于塞入自定义的时间戳等信息。

```
//Android 示例代码
int payloadType = 5;
String msg = "test";
mTXLivePusher.sendSeiMessage(payloadType, msg.getBytes("UTF-8"));
```

常规开源播放器或者网页播放器是不能解析 SEI 消息的，必须使用 LiteAVSDK 中自带的 V2TXLivePlayer 才能解析这些消息：

1. 设置：

```
int payloadType = 5;
mTXLivePlayer.enableReceiveSeiMessage(true, payloadType)
```

2. 当 V2TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 V2TXLivePlayerObserver 中的 onReceiveSeiMessage 回调来接收该消息。

事件处理

事件监听

SDK 通过 V2TXLivePusherObserver 代理来监听推流相关的事件通知和错误通知，详细的事件表和错误码表请参见错误码表。

错误通知

SDK 发现部分严重问题，推流无法继续。

事件 ID	数值	含义说明
V2TXLIVE_ERROR_FAILED	-1	暂未归类的通用错误
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	调用 API 时，传入的参数不合法
V2TXLIVE_ERROR_REFUSED	-3	API 调用被拒绝
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	当前 API 不支持调用
V2TXLIVE_ERROR_INVALID_LICENSE	-5	license 不合法，调用失败
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	请求服务器超时
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	服务器无法处理您的请求

警告事件

SDK 发现部分警告问题，但 WARNING 级别的事件都会触发一些尝试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复。

事件 ID	数值	含义说明
V2TXLIVE_WARNING_NETWORK_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	当前视频播放出现卡顿
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	摄像头打开失败
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	摄像头正在被占用中，可尝试打开其他摄像头
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	麦克风打开失败

事件 ID	数值	含义说明
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	当前系统不支持屏幕分享
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断

录屏推流

最近更新时间：2022-06-14 12:57:06

功能介绍

手机录屏直播，即可以直接把主播的手机画面作为直播源，同时可以叠加摄像头预览，应用于游戏直播、移动端 App 演示等需要手机屏幕画面的场景。腾讯云 LiteAVSDK 通过 V2TXLivePusher 接口提供录屏推流能力。

> 说明：
> 直播中叠加摄像头预览，即通过在手机上添加浮框，显示摄像头预览画面。录屏的时候会把浮框预览画面一并录制下来，达到叠加摄像头预览的效果。

限制说明

- Android 5.0 系统以后开始支持录屏功能。
- 悬浮窗在部分手机和系统上需要通过手动设置打开。

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

对接攻略

步骤1：创建 Pusher 对象

创建一个 **V2TXLivePusher** 对象，我们后面主要用它来完成推流工作。

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(context, V2TXLiveDef.V2TXLive
Mode.TXLiveMode_RTMP);
```

步骤2：启动推流

经过 步骤1 的准备之后，用下面这段代码就可以启动推流了：

```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startMicrophone();
mLivePusher.startScreenCapture();
mLivePusher.startPush(rtmpUrl);
```

- **startScreenCapture** 的作用是启动屏幕录制，由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出提示，允许即可。
- **startPush** 的作用是告诉 LiteAV SDK 音视频流要推到哪个推流 URL 上去。

步骤3：设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数 `(x, y, scale)` 所决定。

- SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。
- `(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`(0.1, 0.1, 0.1)`，那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

```
//设置视频水印
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.w
atermark), 0.03f, 0.015f, 1f);
```

步骤4：推荐的清晰度

调用 V2TXLivePusher 中的 `setVideoQuality` 接口，可以设定观众端的画面清晰度。之所以说是观众端的画面清晰度，是因为主播看到的视频画面是未经编码压缩过的高清原画，不受设置的影响。而 `setVideoQuality` 设定的视频编码器的编码质量，观众端可以感受到画质的差异。详情请参见 设定画面质量。

步骤5：提醒主播“网络不好”

手机连接 Wi-Fi 网络不一定就非常好，如果 Wi-Fi 信号差或者出口带宽很有限，可能网速不如4G，如果主播在推流时遇到网络很差的情况，需要有一个友好的提示，提示主播应当切换网络。

通过 V2TXLivePusherObserver 里的 onWarning 可以捕获 **V2TXLIVE_WARNING_NETWORK_BUSY** 事件，它代表当前主播的网络已经非常糟糕，出现此事件即代表观众端会出现卡顿。此时就可以像上图一样在 UI 上弹出一个“弱网提示”。

```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
showNetBusyTips(); // 显示网络繁忙的提示
}
}
```

步骤6：横竖屏适配

大多数情况下，主播习惯以“竖屏持握”手机进行直播拍摄，观众端看到的也是竖屏分辨率的画面（例如 540 × 960 这样的分辨率）；有时主播也会“横屏持握”手机，这时观众端期望能看到是横屏分辨率的画面（例如 960 × 540 这样的分辨率）。

V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResoluti
onModeLandscape : V2TXLiveVideoResolutionModePortrait);
```

步骤7：结束推流

因为用于推流的 `V2TXLivePusher` 对象同一时刻只能有一个在运行，所以结束推流时要做好清理工作。

```
//结束录屏直播，注意做好清理工作
public void stopPublish() {
mLivePusher.stopScreenCapture();
mLivePusher.setObserver(null);
mLivePusher.stopPush();
}
```

事件处理

事件监听

SDK 通过 V2TXLivePusherObserver 代理来监听推流相关的事件通知和错误通知，详细的事件表和错误码表请参见错误码表。

错误通知

SDK 发现部分严重问题，推流无法继续。

事件 ID	数值	含义说明
V2TXLIVE_ERROR_FAILED	-1	暂未归类的通用错误
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	调用 API 时，传入的参数不合法
V2TXLIVE_ERROR_REFUSED	-3	API 调用被拒绝
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	当前 API 不支持调用
V2TXLIVE_ERROR_INVALID_LICENSE	-5	license 不合法，调用失败

事件 ID	数值	含义说明
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	请求服务器超时
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	服务器无法处理您的请求

警告事件

SDK 发现部分警告问题，但 WARNING 级别的事件都会触发一些尝试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复。

事件 ID	数值	含义说明
V2TXLIVE_WARNING_NETWORK_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	当前视频播放出现卡顿
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	摄像头打开失败
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	摄像头正在被占用中，可尝试打开其他摄像头
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	麦克风打开失败
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	当前系统不支持屏幕分享
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断

Web 推流

最近更新时间：2022-06-14 12:57:38

TXLivePusher 推流 SDK 主要用于视频云的快直播（超低延迟直播）推流，负责将浏览器采集的音视频画面通过 WebRTC 推送到直播服务器。目前支持摄像头推流、屏幕录制推流和本地媒体文件推流。

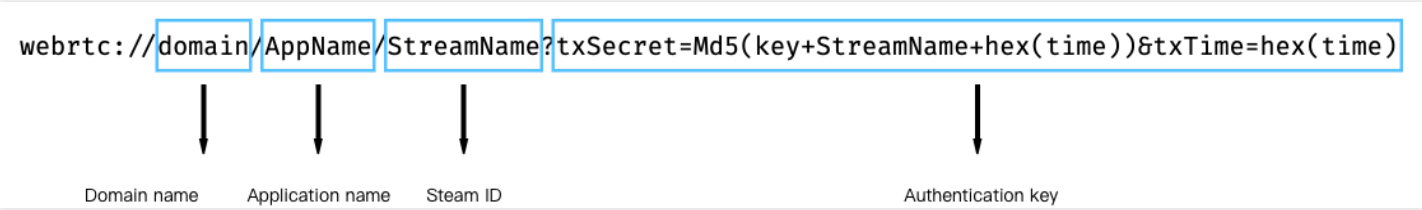
> 注意：
> 使用 WebRTC 协议推流，每个推流域名默认限制**100路并发**推流数，如您需要超过此推流限制，可通过 提交工单 的方式联系我们进行申请。

基础知识

对接前需要了解以下基础知识：

推流地址的拼装

使用腾讯云直播服务时，推流地址需要满足腾讯云标准直播推流 URL 的格式，如下所示，它由四个部分组成：



其中鉴权 Key 部分非必需，如果需要防盗链，请开启推流鉴权，具体使用说明请参见 自主拼装直播 URL 。

浏览器支持

快直播推流基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持。

除此以外，浏览器采集音视频画面的功能在移动端支持较差，例如移动端浏览器不支持屏幕录制，iOS 14.3及以上版本才支持获取用户摄像头设备。因此推流 SDK 主要适用于桌面端浏览器，目前最新版本的 chrome、Firefox 和 Safari 浏览器都是支持快直播推流的。

移动端建议使用 移动直播 SDK 进行推流。

对接攻略

步骤1：页面准备工作

在需要直播推流的页面（桌面端）中引入初始化脚本。

```
<script src="https://imgcache.qq.com/open/qcloud/live/webrtc/js/TXLivePusher-1.0.2.min.js" charset="utf-8"></script>
```

> 说明：
> 需要在 HTML 的 body 部分引入脚本，如果在 head 部分引入会报错。

如果在域名限制区域，可以引入以下链接：

```
<script src="https://cloudcache.tencent-cloud.com/open/qcloud/live/webrtc/js/TXLivePusher-1.0.2.min.js" charset="utf-8"></script>
```

步骤2：在 HTML 中放置容器

在需要展示本地音视频画面的页面位置加入播放器容器，即放一个 div 并命名，例如 id_local_video，本地视频画面都会在容器里渲染。对于容器的大小控制，您可以使用 div 的 css 样式进行控制，示例代码如下：

```
<div id="id_local_video" style="width:100%;height:500px;display:flex;align-items:center;justify-content:center;"></div>
```

步骤3：直播推流

1. **生成推流 SDK 实例：**
   通过全局对象 `TXLivePusher` 生成 SDK 实例，后续操作都是通过实例完成。

   ```
   var livePusher = new TXLivePusher();
   ```

2. **指定本地视频播放器容器：**
   指定本地视频播放器容器 div，浏览器采集到的音视频画面会渲染到这个 div 当中。

   ```
   livePusher.setRenderView('id_local_video');
   ```

> 说明：

调用 `setRenderView` 生成的 video 元素默认有声音，如果需要静音的话，可以直接获取 video 元素进行操作。

```
>document.getElementById('id_local_video').getElementsByTagName('video')[0].muted = true;
```

3. **设置音视频质量：**

采集音视频流之前，先进行音视频质量设置，如果预设的质量参数不满足需求，可以单独进行自定义设置。

```
// 设置视频质量
livePusher.setVideoQuality('720p');
// 设置音频质量
livePusher.setAudioQuality('standard');
// 自定义设置帧率
livePusher.setProperty('setVideoFPS', 25);
```

4. **开始采集流：**

目前支持采集摄像头设备、麦克风设备、屏幕录制和本地媒体文件的流。当音视频流采集成功时，播放器容器中开始播放本地采集到的音视频画面。

```
// 打开摄像头
livePusher.startCamera();
// 打开麦克风
livePusher.startMicrophone();
```

5. **开始推流：**

传入腾讯云快直播推流地址，开始推流。推流地址的格式参考 腾讯云标准直播 URL ，只需要将 RTMP 推流地址前面的 `rtmp://` 替换成 `webrtc://` 即可。

```
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
```

说明：

推流之前要保证已经采集到了音视频流，否则推流接口会调用失败，如果要实现采集到音视频流之后自动推流，可以通过回调事件通知，当收到采集首帧成功的通知后，再进行推流。如果同时采集了视频流和音频

流，需要在视频首帧和音频首帧的采集成功回调通知都收到后再发起推流。

```
>var hasVideo = false;
var hasAudio = false;
var isPush = false;
livePusher.setObserver({
onCaptureFirstAudioFrame: function() {
hasAudio = true;
if (hasVideo && !isPush) {
isPush = true;
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime
=xxx');
}
},
onCaptureFirstVideoFrame: function() {
hasVideo = true;
if (hasAudio && !isPush) {
isPush = true;
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime
=xxx');
}
}
});
```

6. **停止快直播推流：**

```
livePusher.stopPush();
```

7. **停止采集音视频流：**

```
// 关闭摄像头
livePusher.stopCamera();
// 关闭麦克风
livePusher.stopMicrophone();
```

进阶攻略

兼容性

SDK 提供静态方法用于检测浏览器对于 WebRTC 的兼容性。

```
TXLivePusher.checkSupport().then(function(data) {
// 是否支持WebRTC
if (data.isWebRTCSupported) {
console.log('WebRTC Support');
} else {
console.log('WebRTC Not Support');
}
// 是否支持H264编码
if (data.isH264EncodeSupported) {
console.log('H264 Encode Support');
} else {
console.log('H264 Encode Not Support');
}
});
```

回调事件通知

SDK 目前提供了回调事件通知，可以通过设置 Observer 来了解 SDK 内部的状态信息和 WebRTC 相关的数据统计。具体内容请参见 TXLivePusherObserver。

```
livePusher.setObserver({
// 推流警告信息
onWarning: function(code, msg) {
console.log(code, msg);
},
// 推流连接状态
onPushStatusUpdate: function(status, msg) {
console.log(status, msg);
},
// 推流统计数据
onStatisticsUpdate: function(data) {
console.log('video fps is ' + data.video.framesPerSecond);
}
});
```

设备管理

SDK 提供了设备管理实例帮助用户进行获取设备列表、切换设备等操作。

```
var deviceManager = livePusher.getDeviceManager();
// 获取设备列表
deviceManager.getDevicesList().then(function(data) {
```

```
data.forEach(function(device) {
console.log(device.deviceId, device.deviceName);
});
});
// 切换摄像头设备
deviceManager.switchCamera('camera_device_id');
```