移动直播 SDK

微信小程序

产品文档

腾讯云

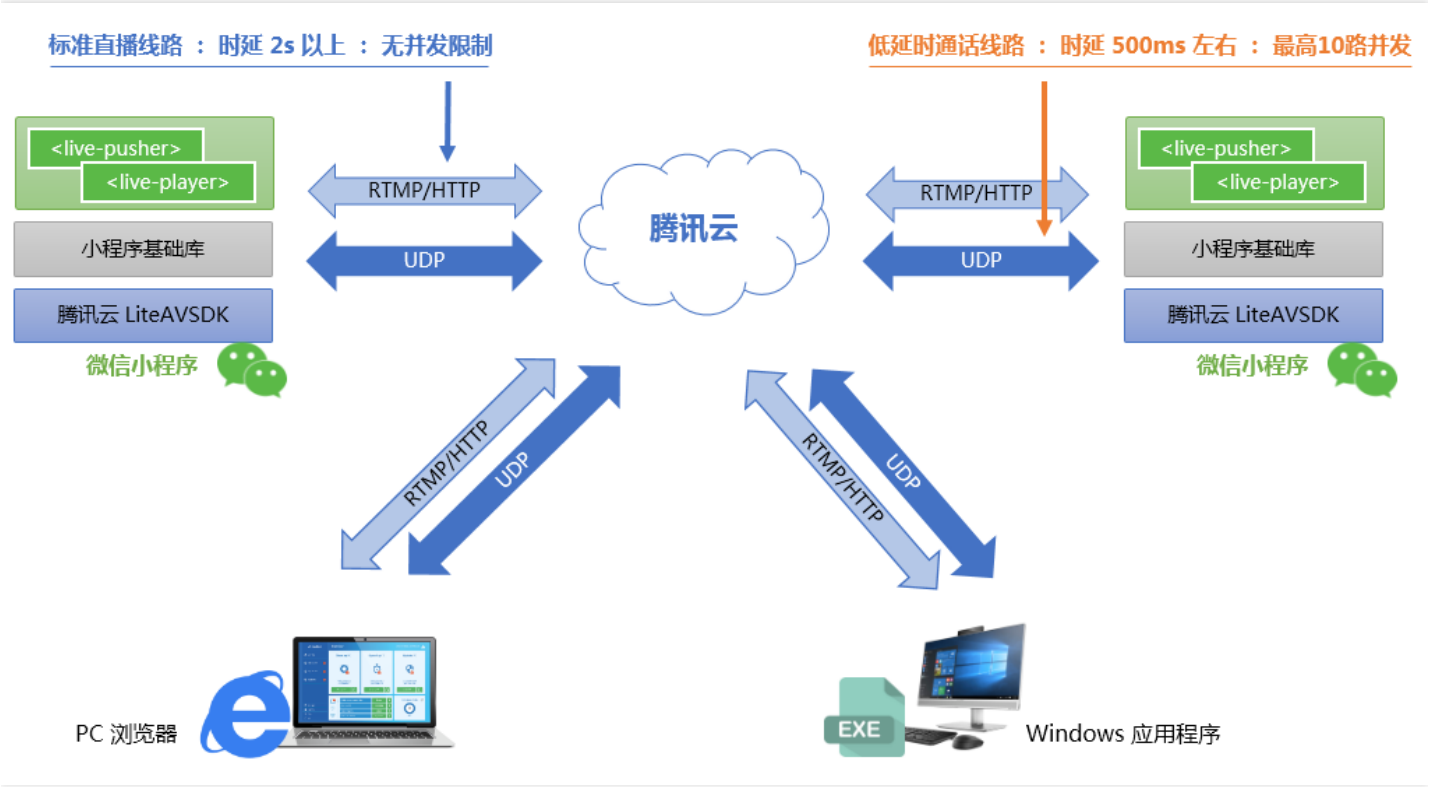
文档目录

微信小程序
小程序音视频解读

最近更新时间：2019-03-26 17:40:11

背景介绍

微信在 2017 下半年的版本中，开始集成移动直播 SDK 到小程序解决方案中，并通过 <live-pusher> 和 <live-player> 两个标签，封装 LiteAVSDK 的 TXLivePusher 和 TXLivePlayer 接口。



两个标签虽然简单，却可以实现绝大多数应用场景下的音视频功能，接下来我将结合一些典型场景为您逐一解读：

产品体验

- **微信端（小程序）**


微信扫一扫，使用小程序
小程序
搜索小程序
附近的小程序
腾讯视频云
小程序开发助手
腾讯行政
冲顶大会
大咖讲学堂
跳一跳
小程序入口
腾讯视频云
以下将展示小程序互动音视频能力，由腾讯云提供技术支持
直播体验室
双人音视频
多人音视频
点播播放器
RTMP推流
直播播放器
低延时播放
更多功能
敬请期待
主功能界面
新的直播间
输入文字内容
双人音视频

- **企业端（PC）**

功能项	小程序组件	PC端体验页面	依赖的云服务	功能描述
手机直播	<live-room>	N/A	直播+云通讯	演示基于小程序的个人直播解决方案
PC 直播	<live-room>	WebEXE	直播+云通讯	演示课堂直播和学生互动的相关功能（需要 PC 端配合）
双人通话	<rtc-room>	WebEXE	直播+云通讯	演示双人视频通话功能，可用于在线客服
多人通话	<rtc-room>	N/A	直播+云通讯	演示多人视频通话功能，可用于临时会议
WebRTC	<webrtc-room>	Chrome	实时音视频	演示小程序和 Chrome 浏览器的互通能力
RTMP推流	<live-pusher>	N/A	直播	演示基础的 RTMP 推流功能
直播播放器	<live-player>	N/A	直播	演示基于 RTMP 和 FLV 协议的直播播放功能

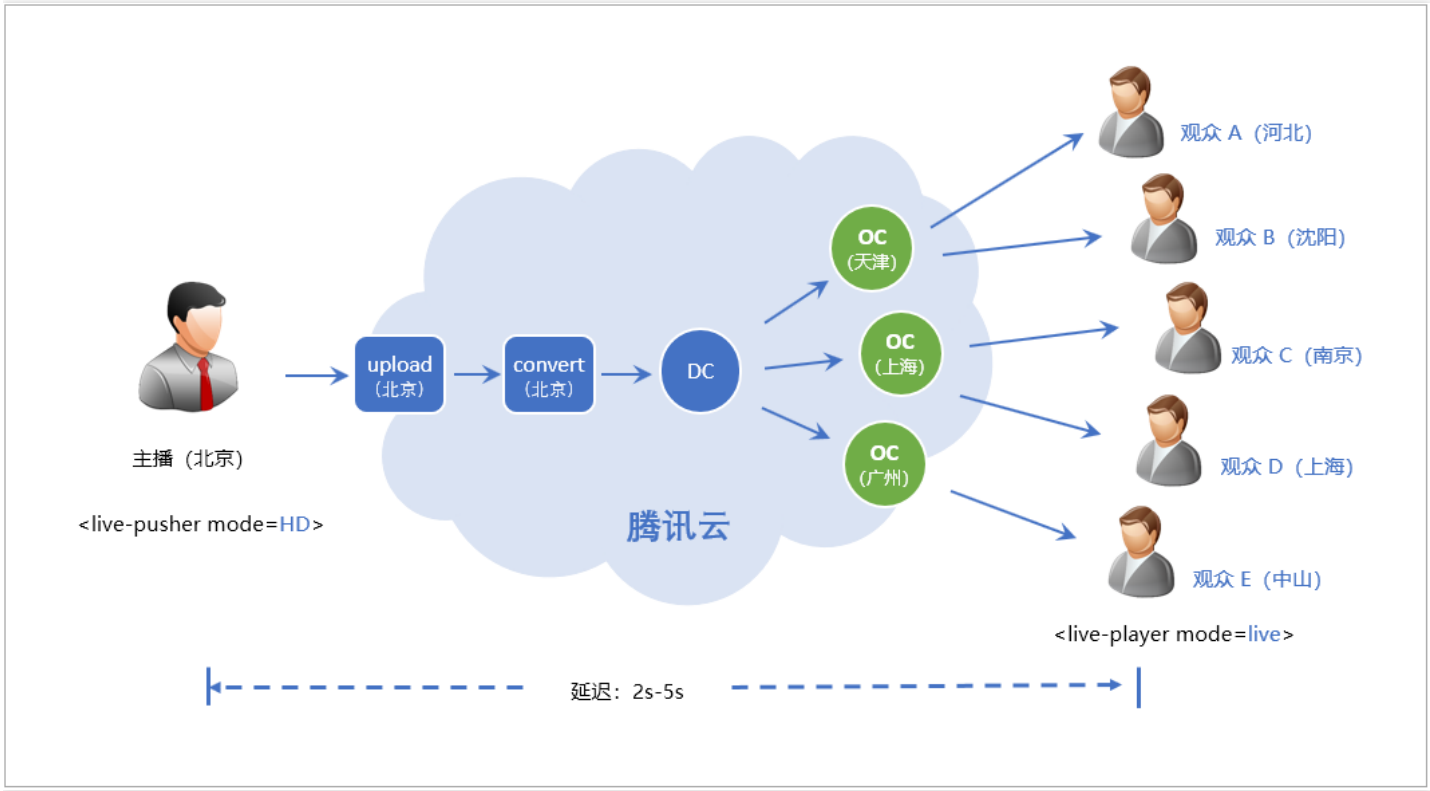
如何开通

- **step1：开通标签使用权限**
  出于政策合规的考虑，微信暂时只允许如下表格中的类目使用 <live-pusher> 和 <live-player> 两个标签。
  符合类目要求的小程序，也需要在小程序管理后台的 “设置 - 接口设置” 中自助开通两个标签的权限。

主类目	子类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医院	问诊、大型健康讲座等直播
金融	银行、信托、基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融	金融产品视频客服理赔、金融产品推广直播等
汽车	汽车预售服务	汽车预售、推广直播
政府主体帐号	-	政府相关工作推广直播、领导讲话直播等
工具	视频客服	不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等

- **step2：开通腾讯云直播服务**
  小程序音视频依赖腾讯云提供的直播（LVB）和云通讯（IM）两项基础服务，您可以单击链接免费开通，其中云通讯服务开通即可使用，直播服务由于涉黄涉政风险较大，需要腾讯云人工审核开通。

标准直播场景：在线培训



场景解读

在线培训是一个非常经典的在线直播场景，您只需要简单的将两个标签组合在一起即可，<live-pusher> 负责将本地画面和声音实时上传到腾讯云，<live-player> 则负责从腾讯云实时拉取视频流并进行播放。

腾讯云在这里的作用就是信号放大器，它负责将来自 <live-pusher> 的一路音视频进行放大，扩散到全国各地，让每一个 <live-player> 都能在离自己比较近的云服务器上拉取到实时且流畅的音视频流。由于原理简单、稳定可靠且支持几百万同时在线的高并发观看，所以从在线教育到体育赛事，从游戏直播到花椒映客，都是基于这种技术实现的。

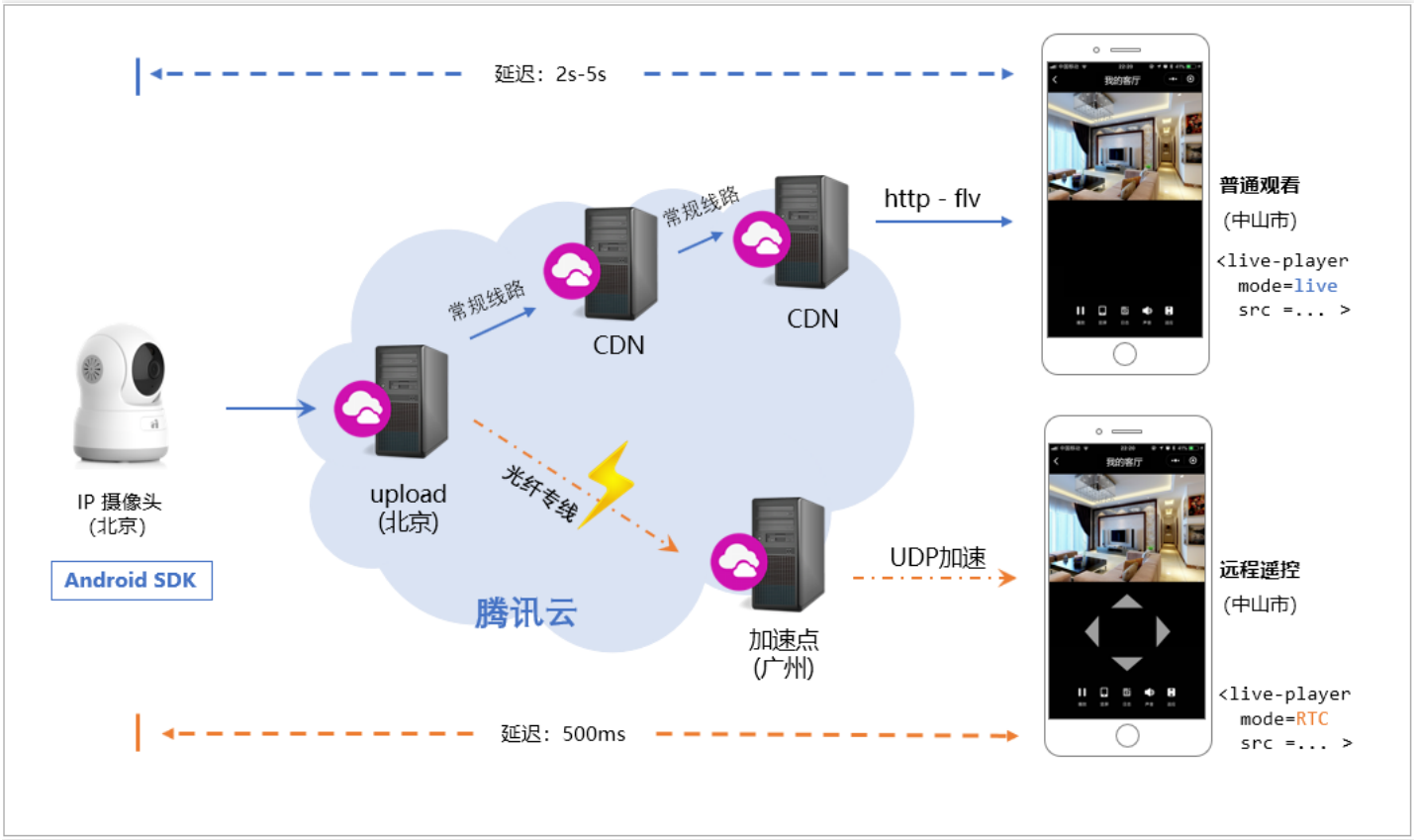
该方案的平均延迟在 2s - 5s，可以通过 <live-player> 标签的 min-cache 和 max-cache 标签进行设置，数值越小，延迟越低，相应的卡顿率也就越高。

对接步骤

- step1. 获取URL：参考 快速获取URL，或者了解如何自行 拼装URL。
- step2. 对接推流：使用 <live-pusher> 标签可以推流到 step1 中获得的 rtmp 推流地址（域名一般为 livepush.mycloud.com），并将 mode 设置为 **HD**。如果这一步失败，可以通过打开 <live-pusher> 中的 debug 功能查看具体问题，或者参考 DOC 排查问题原因。

- step3. 对接播放：使用 <live-player> 标签可以播放 src 指定的 rtmp 或 http-flv 地址 (http-flv是更为推荐的解决方案)，并将 mode 设置为 **live** ，其中 min-cache 和 max-cache 可以分别设置为 1 和 3。

超低延时场景：远程遥控



场景解读

在安防监控的场景里，家用 IPCamera 一般都带有云台旋转的功能，也就是摄像头的指向会跟随远程的遥控进行转动，如果画面延时比较大，那么观看端按下操控按钮到看到画面运动所需要等待的时间就会比较长，这样用户体验就会特别不好。

再比如 2017 非常流行的在线夹娃娃场景，如果远程玩家视频画面的延时非常高，那么远程操控娃娃机就变得不太可能，没有谁能真正抓到娃娃。

要达到这么低的要求，普通的在线直播技术就不再适用了，我们需要使用超低延迟模式，也就是 <live-player> 的 RTC 模式，同时，播放 url 要拼接防盗链签名，以便使用腾讯云的超低延时链路。

对接步骤

- step1. 获取 URL：参考 快速获取URL，或者了解如何自行 拼装URL。

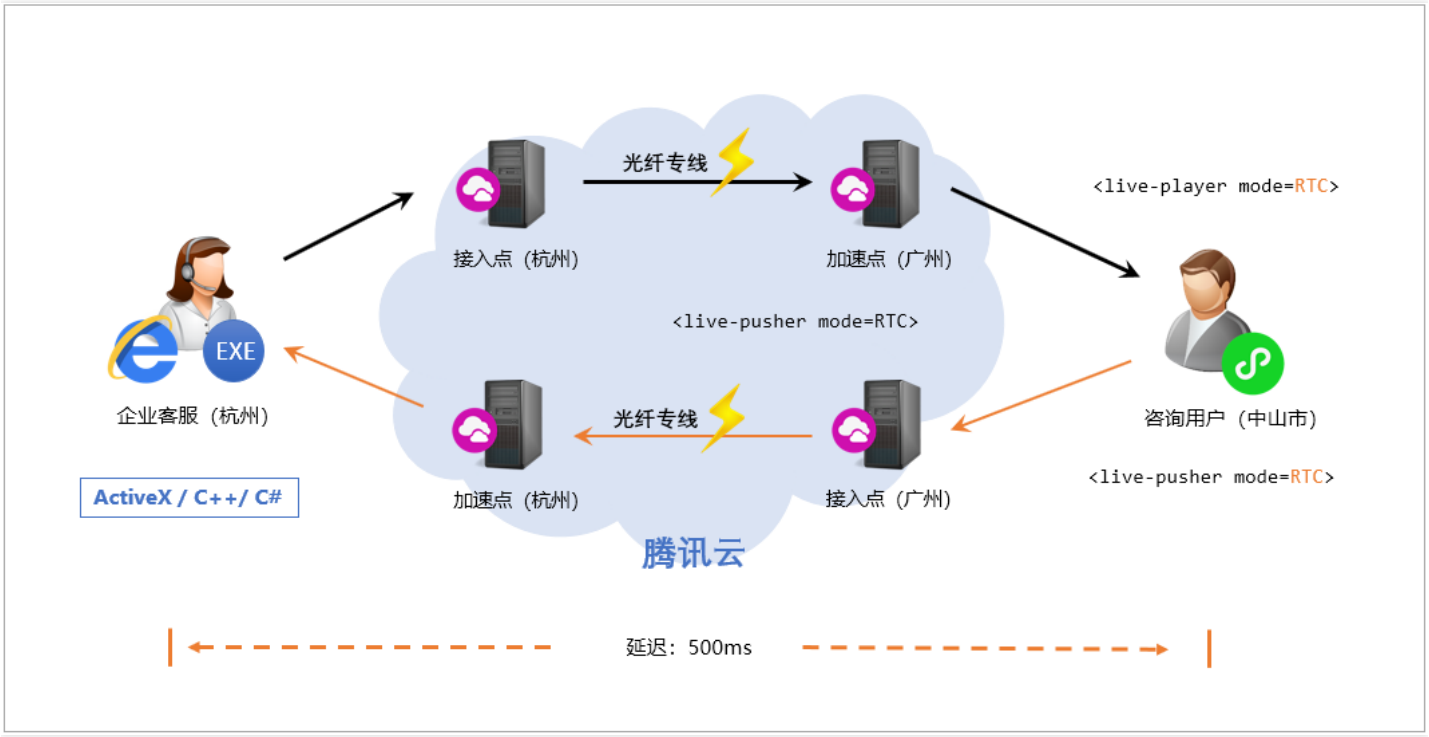
- step2. 对接推流：在设备侧可以使用 Android 版本的 LiteAVSDK(TXLivePusher) 进行推流，如果您需要其它解决方案，也可以通过工单或者 400 电话联系我们。

- step3. 低时延 URL：给普通的 rtmp:// 播放 URL 添加上防盗链签名，就可以将普通的 URL 成低时延 URL，如下：

对比项目	示例	时延
普通直播 URL	rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc	>2s
超低延时 URL	rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc?bizid=bizid&txTime=5FD4431C&txSerect=20e6d865f462dff61ada209d53c71cf9	< 500ms

- step4. RTC 播放：使用 <live-player> 标签，使用 step3 中的低延时 URL 作为其 **src** 属性参数，并将其 mode 设置为 **RTC**，此时 <live-player> 会开启 **延时控制** 功能 和 **UDP加速** 功能。

双向视频通话：视频客服



场景解读

金融开户场景里，银行或者证券公司对开户申请人的身份核实尤为重要，而且要做双录。而开户申请人往往是缘于优惠活动或者广告吸引才有开户意向，这个时候如果要申请人必须安装 App 才能走通整个流程显然非常困难，小程序可以很好地解决 App 安装问题，同时，小程序的正规性也比较容易被用户接受，因此非常适合用于在线金融开户。

同样，保险理赔也是一个比较典型的应用场景，通过小程序的快速安装和启动，可以减少保险公司定损员的出勤率，对于节约运营成本非常有用。

对接步骤（基于原生标签对接）

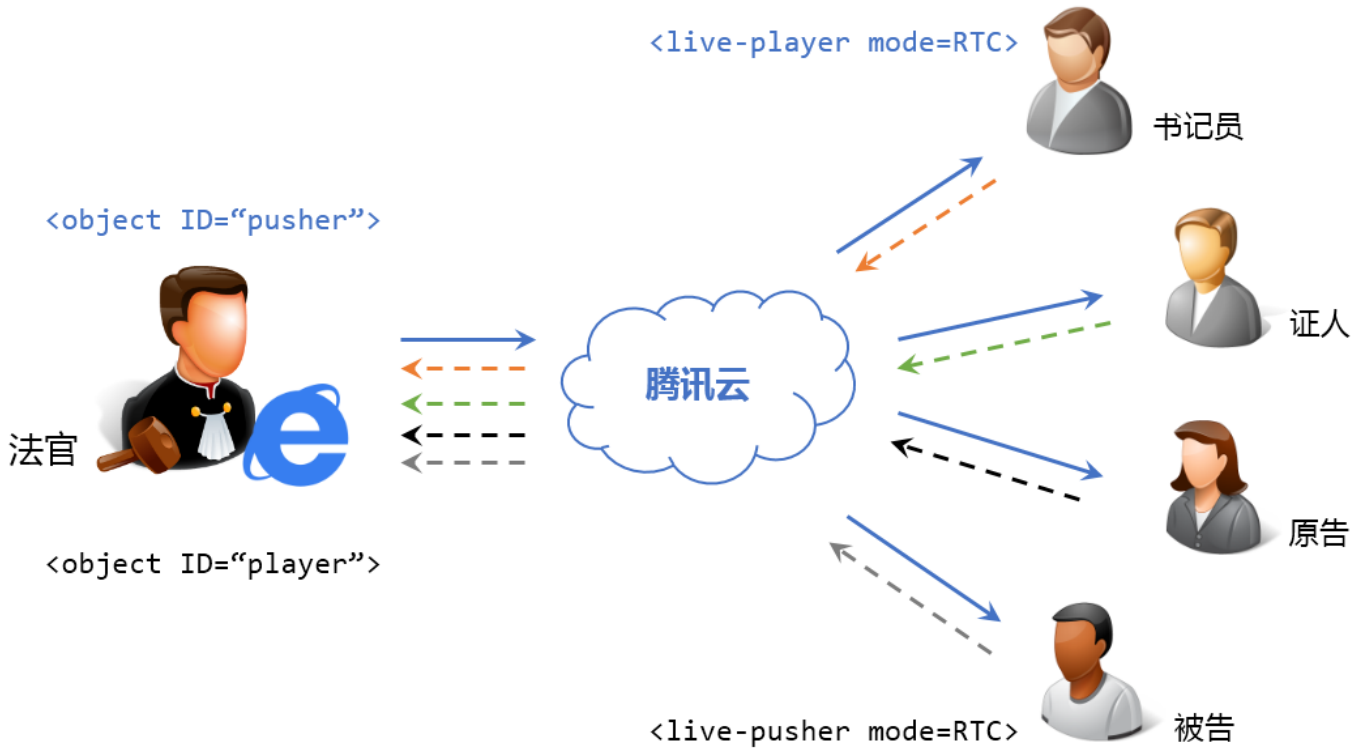
- step1. 准备：首先请阅读 超低延时场景 方案，了解低延时 URL 的获取方式和 RTC 模式的工作原理。
- step2. 客服：推一路视频流到腾讯云，在收到 **1002** 的 onPushEvent 之后，可以将对应的低延时播放地址 urlA 交给 B。由于是客服端，继续使用小程序已经不再合适，您可以选择我们的 IE 、C++ 或者 C# 解决方案。
- step3. 用户：创建一个 <live-player> 标签，mode 设置为 RTC，src 指定为 urlA。
- step4. 用户：创建一个 <live-pusher> 标签，mode 设置为 RTC，并在收到 **1002** 的 onPushEvent 之后，将对应的低延时播放地址 urlB 交给 A。
- step5. 客服：播放低延时模式的 urlB。由于是客服端，继续使用小程序已经不再合适，您可以选择我们的 IE 、C++ 或者 C# 解决方案。

对接步骤（基于 <rtc-room> 实现）

如果您觉得基于原生 <live-pusher> 和 <live-player> 实现的对接方案比较复杂，也可以考虑基于 <rtc-room> 标签进行对接。

- step1：开通：腾讯云直播（LVB）和云通讯（IM）两项基础服务。
- step2：小程序端：使用自定义组件 <rtc-room> 实现视频通话，因为是 1v1 会话模式，所以 <rtc-room> 的 template 可以设定为 **1v1**，当然您也可以自定义 UI 布局。
- step3：Windows：我们同步地提供了多平台的 API ，供您依据自己的项目需要进行选择。

多人视频通话：远程庭审（RTCRoom）



场景解读

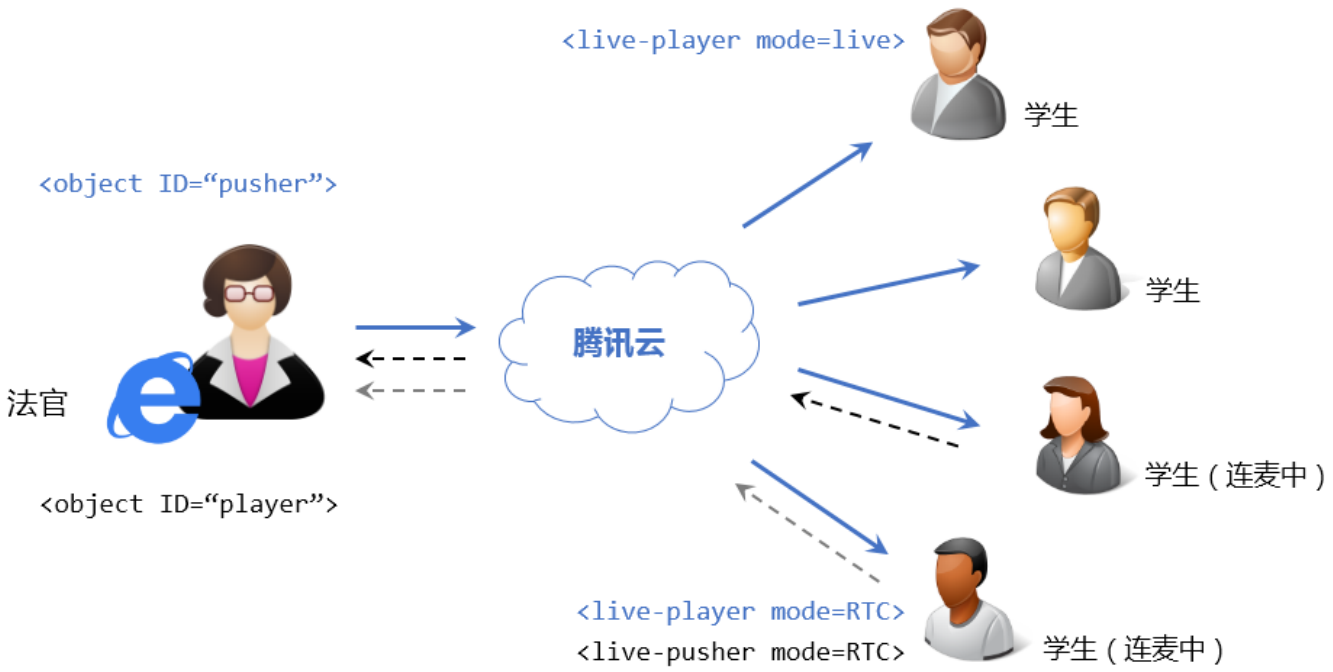
“让信息多跑路，让群众少跑腿”，是中国实行“互联网 + 政务服务”改革的重要目标。远程庭审就是一个典型的应用场景。

小程序的多人音视频非常适合搭建远程庭审方案：一方面，小程序的安装和启动都没有什么心理障碍，这让不方便直接出庭的当事人或者证人更容易参与到庭审案件中。另一方面，小程序本身的正规性也让使用者比较放心。

对接步骤

- step1：开通腾讯云直播（LVB）和云通讯（IM）两项基础服务。
- step2：使用自定义组件 <rtc-room> 实现视频通话，template 可以选择我们预定义的几种模式，当然您也可以自定义 UI 布局。
- step3：Windows：我们同步地提供了多平台的 API ，供您依据自己的项目需要进行选择。

混合场景：直播+连麦（LiveRoom）



场景解读

直播场景，既要支持上万人的并发观看，又要支持观众跟主播可以发起连麦互动，那就需要混合场景解决方案了。混合场景就是在标准直播场景的基础上加入连麦通话能力。

常见的应用场景是互动课堂，互动课堂中的小班课程，一堂课也就几个人，用 <rtc-room> 就可以解决问题了；但如果是大班课，那就需要使用 <live-room> 混合方案了。

腾讯云的 <live-room> 混合方案还有外带的 PPT 白板组件支持，该组件基于小程序的 Canvas 控件构建而成。

对接步骤

- step1：开通：腾讯云直播（LVB）和云通讯（IM）两项基础服务。
- step2：小程序端：使用自定义组件 <live-room> 实现直播 + 连麦功能，template 可以选择我们预定义的几种模式，当然您也可以自定义 UI 布局。
- step3：Windows：我们同步地提供了多平台的 API ，供您依据自己的项目需要进行选择。

录制指引

腾讯云支持小程序音视频解决方案的全程服务端录制，您可以按如下步骤开启云端录制功能。在不开启录制的情况下，腾讯视频云的各个服务器节点只负责中转音视频数据，并不对数据进行其他处理。

- step1： 开通 腾讯云点播服务。
- step2：进入直播控制台（小程序音视频流媒体是基于直播服务构建的），在【接入管理>>接入配置>>直播录制】中，开启录制功能。（注意：这里说的录制费用是按并发收费的，不是每一路都收费）

- step3：在点播的视频管理界面中，您可以看到这些录制的文件，您也可以通过点播服务的 REST API 获取到这些文件。

源码调试

最近更新时间：2019-05-09 12:06:33

Demo 体验

升级微信到最新版本，发现页卡 => 小程序 => 搜索“腾讯视频云”，即可打开小程序 Demo：

功能项	小程序组件	PC端体验页面	依赖的云服务	功能描述
手机直播	<live-room>	N/A	直播 + 云通讯	演示基于小程序的个人直播解决方案
PC 直播	<live-room>	WebEXE	直播 + 云通讯	演示课堂直播和学生互动的相关功能（需要 PC 端配合）
双人通话	<rtc-room>	WebEXE	直播 + 云通讯	演示双人视频通话功能，可用于在线客服
多人通话	<rtc-room>	N/A	直播 + 云通讯	演示多人视频通话功能，可用于临时会议
WebRTC	<webrtc-room>	Chrome	实时音视频	演示小程序和 Chrome 浏览器的互通能力
RTMP推流	<live-pusher>	N/A	直播	演示基础的 RTMP 推流功能
直播播放器	<live-player>	N/A	直播	演示基于 RTMP 和 FLV 协议的直播播放功能

注册小程序并开通相关接口

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

个人账号和企业账号的小程序暂时只开放如下表格中的类目：

主类目	子类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医院	问诊、大型健康讲座等直播
金融	银行、信托、基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融	金融产品视频客服理赔、金融产品推广直播等

主类目	子类目	小程序内容场景
汽车	汽车预售服务	汽车预售、推广直播
政府主体帐号	-	政府相关工作推广直播、领导讲话直播等
工具	视频客服	不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等

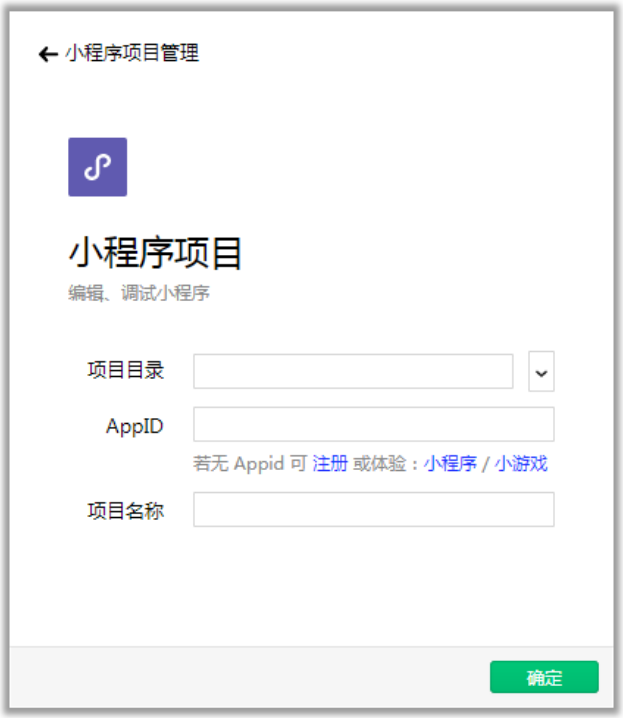
打开 微信公众平台 注册并登录小程序，并在小程序管理后台的 “设置 - 接口设置” 中自助开通该组件权限，如下图所示：

⚠ 注意：

- 该接口对小程序的类目设置有限制要求，详情请参考 小程序开发组件 live - player。
- 如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

安装微信小程序开发工具

下载并安装最新版本的 微信开发者工具，使用小程序绑定的微信号扫码登录开发者工具。



获取 Demo 源码并调试

- step1: 访问 SDK + Demo，获取小程序 Demo 源码。
- step2: 打开安装的微信开发者工具，单击【小程序项目】。
- step3: 输入小程序 AppID，项目目录选择上一步下载下来的代码目录（ **注意：** 目录请选择**根目录**，根目录包含有 `project.config.json` 文件，请不要只选择 `wxlite` 目录！），单击确定创建小程序项目。
- step4: 再次单击【确定】进入开发者工具。
- step5: 请使用手机进行测试，直接扫描开发者工具预览生成的二维码进入。
- step6: 开启调试模式，体验和调试内部功能。开启调试可以跳过把这些域名加入小程序白名单的工作。

Demo 访问的测试地址

Demo 小程序会访问如下表格中的测试服务器地址，这些服务器使用的云服务是我们为大家提供的一个体验账号，平时很多客户都会在上面做测试。如果您希望使用自己的后台服务器，以免被其他客户打扰，请关注文档后一节内容：

- **<live-room> 和 <rtc-room> 相关 Demo 需要访问如下地址：**

URL	对应的服务器地址	服务器的功能描述
https://webim.tim.qq.com	IM 云通讯后台服务地址	用于支持小程序里面的一些消息通讯功能
https://room.qcloud.com	RoomService 后台服务地址	RoomService 是用于支撑 <rtc-room>（视频通话）和 <live-room>（直播连麦）的房间管理逻辑

- **<webrtc-room> 相关 Demo 需要访问如下地址：**

URL	对应的服务器地址	服务器的功能描述

URL	对应的服务器地址	服务器的功能描述
https://webim.tim.qq.com	IM 云通讯后台服务地址	用于支持小程序里面的一些消息通讯功能
https://official.opensso.tencent-cloud.com/v4/openim/jsonvideoapp	WebRTC 测试后台	用于请求进入<webrtc-room> 所需的 userSig 和 privateMapKey
https://xzb.qcloud.com/webrtc/weapp/webrtc_room	WebRTC 房间列表后台	一个简单的房间列表功能，方便 Demo 的测试和使用。

搭建自己的账号和后台服务器

这部分我们将介绍如何将 Demo 默认的测试用服务器地址，换成您自己的服务器，这样一来，您就可以使用自己的腾讯云账号实现上述功能，同时也便于您进行二次开发。

1. 搭建 <webrtc-room> 的服务器

1.1 这个服务器能做什么？

- 单击 Demo 里的互动课堂 **<webrtc-room>** 功能，您会看到一个房间列表，这个房间列表是怎么实现的呢？
- 在看到视频房间列表以后，如果您要创建一个视频房间，或者进入一个其他人建好的视频房间，就需要为 <webrtc-room> 所对应的几个属性（ sdkAppID 、 userID 、 userSig 、 roomID 和 privateMapKey ）传递合法的参数值，这几个参数值怎么获取呢？

1.2 这个服务器要怎么搭建？

- 下载 webrtc_server ，这是一份 Java 版本的实现，根据 README.md 中的说明就可以了解怎么使用这份源码。

1.3 服务器建好了我怎么用？

- 小程序 源码中，将 wxlite/config.js 文件中的 webrtcServerUrl 修改成：

```
https://您自己的域名/webrtc/weapp/webrtc_room
```

- 小程序实现 WebRTC 能力肯定是为了跟 Chrome 浏览器进行视频通话，浏览器端的源代码可以单击 Chrome(src) 下载到，将 component/WebRTCRoom.js 文件中的 serverDomain 修改成：

```
https://您自己的域名/webrtc/weapp/webrtc_room
```

2. 搭建 <live-room> 和 <rtc-room> 的服务器

2.1 这个服务器能做什么？

- <live-room>（用于直播连麦）和 <rtc-room>（用于视频通话）都是基于腾讯云 LVB 和 IM 两个基础服务实现的扩展功能，需要一个叫做 RoomService 的后台组件配合才能运行。

2.2 这个服务器要怎么搭建？

- 下载 RoomService 的 Java 版本源代码，根据 README.md 中的说明就可以了解怎么使用这份源码。

2.3 服务器建好了我怎么用？

- 小程序 源码中，将 `wxlite/config.js` 文件中的 `serverUrl` 和 `roomServiceUrl` 修改成：

```
https://您自己的域名/roomservice/
```

- 小程序如果使用 <live-room> 和 <rtc-room> 两个标签，在 PC 端就不能用 Chrome 浏览器配对了，需要改用 WebEXE 混合解决方案。将 GitHub(WebEXE) 源码中 liveroom.html、double.html文件中的 `RoomServerDomain` 修改成:

```
https://您自己的域名/roomservice/
```

3. Wafer 零成本服务器部署方案（Node.js）

如果您是一位资深的 Web 前端工程师，暂时找不到合适的服务器，但又想快速拥有自己的调试后台，可以使用腾讯云的 Wafer 功能进行零成本的一键部署方案（Wafer 只支持 Node.js 语言的后台代码），您需要做的只是：

- step1: 下载 小程序 源码。
- step2: 根据 一键部署指引 完成部署。
- step3: 将 GitHub(WebEXE) 源码中 liveroom.html、double.html 文件中的 `RoomServerDomain` 修改成:

```
https://您自己的域名/roomservice/
```

微信端
live-pusher 标签

最近更新时间：2019-03-26 17:39:33

<live-pusher> 是小程序内部用于支持音视频上行能力的功能标签，本文主要介绍该标签的使用方法。

版本支持

- 微信 App iOS 最低版本要求：6.5.21
- 微信 App Android 最低版本要求：6.5.19
- 小程序基础库最低版本要求：1.7.0

> 通过 wx.getSystemInfo 可以获取当前基础库版本信息

使用限制

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 个人账号和企业账号的小程序暂时只开放如下表格中的类目：

主类目	子类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医院	问诊、大型健康讲座等直播
金融	银行、信托、基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融	金融产品视频客服理赔、金融产品推广直播等
汽车	汽车预售服务	汽车预售、推广直播

主类目	子类目	小程序内容场景
政府主体帐号	-	政府相关工作推广直播、领导讲话直播等
工具	视频客服	不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等

- 符合类目要求的小程序，需要在小程序管理后台的 “设置 - 接口设置” 中自助开通该组件权限，如下图所示：

注意：如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

属性定义

属性名	类型	默认值	说明
url	String		用于音视频上行的推流URL
mode	String	RTC	SD, HD, FHD, RTC
autopush	Boolean	false	是否自动启动推流
muted	Boolean	false	是否静音

属性名	类型	默认值	说明
enable-camera	Boolean	true	开启\关闭摄像头
auto-focus	Boolean	true	手动\自动对焦
orientation	String	vertical	vertical, horizontal
beauty	Number	0	美颜指数，取值 0 - 9，数值越大效果越明显
whiteness	Number	0	美白指数，取值 0 - 9，数值越大效果越明显
aspect	String	9:16	3:4, 9:16
zoom	Boolean	false	是否正常焦距，true 表示将摄像头放大
device-position	String	front	front 前置摄像头，back 后置摄像头
min-bitrate	Number	200	最小码率，该数值决定了画面最差的清晰度表现
max-bitrate	Number	1000	最大码率，该数值决定了画面最好的清晰度表现
audio-quality	String	low	low 适合语音通话, high 代表高音质
waiting-image	String		当微信切到后台时的垫片图片
waiting-image-hash	String		当微信切到后台时的垫片图片的校验值
background-mute	Boolean	false	当微信切到后台时是否禁用声音采集
bindstatechange	String		用于指定一个 javascript 函数来接收音视频事件
debug	Boolean	false	是否开启调试模式

示例代码

```
<view id='video-box'>
<live-pusher
id="pusher"
mode="RTC"
url="{{pusher.push_url}}"
autopush='true'
bindstatechange="onPush">
</live-pusher>
</view>
```

属性详解

- **url**
  用于音视频上行的推流URL，以 rtmp:// 协议前缀打头，腾讯云推流 URL 的获取方法见 快速获取URL 文档。

  > 小程序内部使用的 RTMP 协议是支持 UDP 加速的版本，在同样网络条件下，UDP 版本的 RTMP 会比开源版本的有更好的上行速度和抗抖动能力。

- **mode**
  SD、HD 和 FHD 主要用于直播类场景，比如赛事直播、在线教育、远程培训等等。SD、HD 和 FHD 分别对应三种默认的清晰度。该模式下，小程序会更加注重清晰度和观看的流畅性，不会过分强调低延迟，也不会为了延迟牺牲画质和流畅性。

  RTC 则主要用于双向视频通话或多人视频通话场景，比如金融开会、在线客服、车险定损、培训会议等。该模式下，小程序会更加注重降低点到点的时延，也会优先保证声音的质量，在必要的时候会对画面清晰度和画面的流畅性进行一定的缩水。

- **orientation 和 aspect**
  横屏（horizontal）模式还是竖屏（vertical）模式，默认是竖屏模式，即 home 键朝下。这时，小程序推出的画面的宽高比是 3：4 或者 9 : 16 这两种竖屏宽高比的画面，也就是宽 < 高。如果改成横屏模式，小程序推出的画面宽高比即变为 4 : 3 或者 16 : 9 这种横屏宽高比的画面，也就是宽 > 高。

  具体的宽高比是有 aspect 决定的 ，默认是 9: 16， 也可以支持 3：4。 这是在 orientation 的属性值为 vertical 的情况下。如果 orientation 的属性值为 horizontal，那么 3：4 的效果等价于 4：3， 9：16 的效果等价于 16：9。

- **min-bitrate 和 max-bitrate**
  这里首先要科普一个概念 —— 视频码率，指视频编码器每秒钟输出的视频数据的多少。在视频分辨率确定的情况下，视频码率越高，即每秒钟输出的数据越多，相应的画质也就越好。

  所以 min-bitrate 和 max-bitrate 这两个属性，分别用于决定输出画面的最低清晰度和最高清晰度。这两个数值并非越大越好，因为用户的网络上行不是无限好的。但也不是越小越好，因为实际应用场景中，清晰与否是用户衡量产品体验的一个重要指标。具体的数值设定我们会在 **“参数设置”** 部分详细介绍。

小程序内部会自动处理好分辨率和码率的关系，比如 2Mbps 的码率，小程序会选择 720p 的分辨率进行匹配，而 300kbps 的码率下，小程序则会选择较低的分辨率来提高编码效率。所以您只需要关注 min-bitrate 和 max-bitrate 这一对参数就可以掌控画质了

- **waiting-image 和 waiting-image-hash**
  出于用户隐私的考虑，在微信切到后台以后，小程序希望停止摄像头的画面采集。但是对于另一端的用户而言，画面会变成黑屏或者冻屏（停留在最后一帧），这种体验是非常差的。为了解决这个问题，我们引入了 waiting-image 属性，您可以设置一张有 “稍候” 含义的图片（waiting-image 是该图片的 url，waiting-image-hash 则是该图片对应的 md5 校验值）。当微信切到后台以后，小程序会使用该图片作为摄像头画面的替代，以极低的流量占用维持视频流 3 分钟时间。

- **debug**
  调试音视频相关功能，如果没有很好的工具会是一个噩梦，所以小程序为 live-pusher 标签支持了 debug 模式，开始 debug 模式之后，原本用于渲染视频画面的窗口上，会显示一个半透明的 log 窗口，用于展示各项音视频指标和事件，降低您调试相关功能的难度，具体使用方法我们在 FAQ 中有详细说明。

参数设置

这么多参数，具体要怎样设置才比较合适呢？我们给出如下建议值：

场景	mode	min-bitrate	max-bitrate	audio-quality	说明
标清直播	SD	300kbps	800kbps	high	窄带场景，比如户外或者网络不稳定的情况下适用
高清直播	HD	600kbps	1200kbps	high	目前主流的APP所采用的参数设定，普通直播场景推荐使用这一档
超清直播	FHD	600kbps	1800kbps	high	对清晰度要求比较苛刻的场景，普通手机观看使用 HD 即可
视频客服（用户）	RTC	200kbps	500kbps	high	这是一种声音为主，画面为辅的场景，所以画质不要设置的太高
车险定损（车主）	RTC	200kbps	1200kbps	high	由于可能要看车况详情，画质上限会设置的高一些
多人会议（主讲）	RTC	200kbps	1000kbps	high	主讲人画质可以适当高一些，参与的质量可以设置的低一些

场景	mode	min-bitrate	max-bitrate	audio-quality	说明
多人会议（参与）	RTC	150kbps	300kbps	low	作为会议参与者，不需要太高的画质和音质

如果不是对带宽特别没有信心的应用场景，audio-quality选项请不要选择 low，其音质和延迟感都要会比 high 模式差很多。

对象操作

- **wx.createLivePusherContext()**
  通过 wx.createLivePusherContext() 可以将 <live-pusher> 标签和 javascript 对象关联起来，之后即可操作该对象。

- **start**
  开始推流，如果 <live-pusher> 的 autopush 属性设置为 false（默认值），那么就可以使用 start 来手动开始推流。

- **stop**
  停止推流。

- **pause**
  暂停推流。

- **resume**
  回复推流，请与 pause 操作配对使用。

- **switchCamera**
  切换前后摄像头

- **snapshot**
  推流截图，截图大小跟组件的大小一致。截图成功图片的临时路径为ret.tempImagePath

```
var pusher = wx.createLivePusherContext('pusher');
pusher.start({
```

```
success: function(ret){
console.log('start push success!')
}
fail: function(){
console.log('start push failed!')
}
complete: function(){
console.log('start push complete!')
}
});
```

内部事件

通过 **<live-pusher>** 标签的 **bindstatechange** 属性可以绑定一个事件处理函数，该函数可以监听推流模块的内部事件和异常通知。

1. 常规事件

code	事件定义	含义说明
1001	PUSH_EVT_CONNECT_SUCC	已经成功连接到云端服务器
1002	PUSH_EVT_PUSH_BEGIN	与服务器握手完毕，一切正常，准备开始上行推流
1003	PUSH_EVT_OPEN_CAMERA_SUCC	已成功启动摄像头，摄像头被占用或者被限制权限的情况下无法打开

2. 严重错误

code	事件定义	含义说明
-1301	PUSH_ERR_OPEN_CAMERA_FAIL	打开摄像头失败
-1302	PUSH_ERR_OPEN_MIC_FAIL	打开麦克风失败
-1303	PUSH_ERR_VIDEO_ENCODE_FAIL	视频编码失败
-1304	PUSH_ERR_AUDIO_ENCODE_FAIL	音频编码失败
-1305	PUSH_ERR_UNSUPPORTED_RESOLUTION	不支持的视频分辨率
-1306	PUSH_ERR_UNSUPPORTED_SAMPLERATE	不支持的音频采样率

code	事件定义	含义说明
-1307	PUSH_ERR_NET_DISCONNECT	网络断连,且经三次抢救无效，可以放弃治疗，更多重试请自行重启推流

3. 警告事件

内部警告并非不可恢复的错误，小程序内部的音视频 SDK 会启动相应的恢复措施，警告的目的主要用于提示开发者或者最终用户，比如：

- **PUSH_WARNING_NET_BUSY**
  上行网速不给力，建议提示用户改善当前的网络环境，比如让用户离家里的路由器近一点，或者切到 Wi-Fi 环境下再使用。

- **PUSH_WARNING_SERVER_DISCONNECT**
  请求被后台拒绝了，出现这个问题一般是由于 URL 里的 txSecret 计算错了，或者是 URL 被其他人占用了（跟播放不同，一个推流 URL 同时只能有一个用户使用）。

- **PUSH_WARNING_HANDUP_STOP**
  当用户单击小程序右上角的圆圈或者返回按钮时，微信会将小程序挂起，此时 <live-pusher> 会抛出 5000 这个事件。

code	事件定义	含义说明
1101	PUSH_WARNING_NET_BUSY	上行网速不够用，建议提示用户改善当前的网络环境
1102	PUSH_WARNING_RECONNECT	网络断连，已启动重连流程 (重试失败超过三次会放弃)
1103	PUSH_WARNING_HW_ACCELERATION_FAIL	硬编码启动失败，自动切换到软编码
1107	PUSH_WARNING_SWITCH_SWENC	由于机器性能问题，自动切换到硬件编码
3001	PUSH_WARNING_DNS_FAIL	DNS 解析失败，启动重试流程
3002	PUSH_WARNING_SEVER_CONN_FAIL	服务器连接失败，启动重试流程
3003	PUSH_WARNING_SHAKE_FAIL	服务器握手失败，启动重试流程
3004	PUSH_WARNING_SERVER_DISCONNECT	服务器主动断开连接，启动重试流程
3005	PUSH_WARNING_SERVER_DISCONNECT	socket 链路异常断开 ，启动重试流程

code	事件定义	含义说明
5000	PUSH_WARNING_HANDUP_STOP	小程序被用户挂起

4. 示例代码

```
Page({
onPush: function(ret) {
if(ret.detail.code == 1002) {
console.log('推流成功了',ret);
}
},

/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
//...
}
})
```

live-player 标签

最近更新时间：2019-03-26 17:38:56

<live-player> 是小程序内部用于支持音视频下行（播放）能力的功能标签，本文主要介绍该标签的使用方法。

版本支持

- 微信 App iOS 最低版本要求：6.5.21
- 微信 App Android 最低版本要求：6.5.19
- 小程序基础库最低版本要求：1.7.0

> 通过 wx.getSystemInfo 可以获取当前基础库版本信息

使用限制

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 个人账号和企业账号的小程序暂时只开放如下表格中的类目：

主类目	子类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医院	问诊、大型健康讲座等直播
金融	银行、信托、基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融	金融产品视频客服理赔、金融产品推广直播等
汽车	汽车预售服务	汽车预售、推广直播

主类目	子类目	小程序内容场景
政府主体帐号	-	政府相关工作推广直播、领导讲话直播等
工具	视频客服	不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等

- 符合类目要求的小程序，需要在小程序管理后台的 “设置 - 接口设置” 中自助开通该组件权限，如下图所示：

注意：如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

属性定义

属性名	类型	默认值	说明
src	String		用于音视频下行的播放URL，支持rtmp、flv 等协议
mode	String	live	live, RTC
autoplay	Boolean	false	是否自动播放
muted	Boolean	false	是否静音

属性名	类型	默认值	说明
orientation	String	vertical	vertical, horizontal
object-fit	String	contain	contain, fillCrop
background-mute	Boolean	false	当微信切到后台时，是否关闭播放声音
min-cache	Number	1	最小缓冲延迟， 单位：秒
max-cache	Number	3	最大缓冲延迟， 单位：秒
bindstatechange	EventHandler		用于指定一个javascript函数来接受播放器事件
bindfullscreenchange	EventHandler		用于指定一个javascript函数来接受全屏事件
debug	Boolean	false	是否开启调试模式

示例代码

```
<view id='video-box'>
<live-player
wx:for="{{player}}"
id="player_{{index}}"
mode="RTC"
object-fit="fillCrop"
src="{{item.playUrl}}"
autoplay='true'
bindstatechange="onPlay">
</live-player>
</view>
```

超低时延

<live-player> 的 RTC 模式支持 500ms 以内的超低时延链路，可以应用在视频通话和远程遥控等场景中，要使用超低时延播放，需要注意如下几点：

（1）推流端如果是微信小程序，请使用 <live-pusher> 的 RTC 模式。

（2）推流端如果是 iOS 或者 Android SDK，请使用 setVideoQuality 的 MAIN_PUBLISHER 模式。

（3）推流端如果是 Windows，请不要使用 OBS，延时太高，可以使用我们的 Windows SDK。

（4）<live-player> 的 min-cache 和 max-cache 请不要自行设置，使用默认值。
（5）播放地址请使用超低延时播放地址，也就是带了防盗链签名的 rtmp:// 地址，如下：

对比项目	示例	时延
普通直播 URL	rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc	>2s
超低延时 URL	rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc?bizid=bizid&txTime=5FD4431C&txSerect=20e6d865f462dff61ada209d53c71cf9	< 500ms

属性详解

- **src**
  用于音视频下行的播放 URL，支持 rtmp 协议（URL 以 “rtmp://” 打头）和 flv 协议（URL 以 “http://” 打头且以 “.flv” 结尾），腾讯云推流 URL 的获取方法见 DOC。

  > <live-player> 标签是不支持 HLS(m3u8) 协议的，因为 <video> 已经支持 HLS(m3u8) 播放协议了。但直播观看不推荐使用 HLS(m3u8) 协议，延迟要比 rtmp 和 flv 协议高一个数量级。

- **mode**
  live 模式主要用于直播类场景，比如赛事直播、在线教育、远程培训等等。该模式下，小程序内部的模块会优先保证观看体验的流畅，通过调整 min-cache 和 max-cache 属性，您可以调节观众（播放）端所感受到的时间延迟的大小，文档下面会详细介绍这两个参数。

  RTC 则主要用于双向视频通话或多人视频通话场景，比如金融开会、在线客服、车险定损、培训会议 等等。在此模式下，对 min-cache 和 max-cache 的设置不会起作用，因为小程序内部会自动将延迟控制在一个很低的水平（500ms 左右）。

- **min-cache 和 max-cache**
  这两个参数分别用于指定观看端的最小缓冲时间和最大缓冲时间。所谓**缓冲时间**，是指播放器为了缓解网络波动对观看流畅度的影响而引入的一个“蓄水池”，当来自网络的数据包出现卡顿甚至停滞的时候，“蓄水池”里的紧急用水可以让播放器还能坚持一小段时间，只要在这个短暂的时间内网速恢复正常，播放器就可以源源不断地渲染出流畅而平滑的视频画面。

  “蓄水池”里的水越多，抗网络波动的能力就越强，但代价就是观众端的延迟就越大，所以要在不同的场景下，使用不同的配置来达到体验上的平衡：

- 码率比较低（1Mbps 左右，画面以人物为主）的直播流，min-cache = 1，max-cache = 3 较合适。
  - 码率比较高（2-3Mbps 的高清游戏画面为主）的直播流，min-cache = 3，max-cache = 5 较合适。

RTC 模式下这两个参数是无效的。

- **orientation**
  画面渲染角度，horizontal 代表是原始画面方向，vertical 代表向右旋转 90度。

- **object-fit**
  画面填充模式，contain 代表把画面显示完成，但如果视频画面的宽高比和 <live-player> 标签的宽高比不一致，那么您将看到黑边。fillCrop 代表把屏幕全部撑满，但如果视频画面的宽高比和 <live-player> 标签的宽高比不一致，那么画面中多余的部分会被裁剪掉。

- **background-mute**
  微信切到后台以后是否继续播放声音，用于避免锁屏对于当前小程序正在播放的视频内容的影响。

- **sound-mode**
  设置播放模式，可设值为: ear与speaker，ear代表使用听筒播放， speaker代表使用扬声器。默认为扬声器

- **debug**
  调试音视频相关功能，如果没有很好的工具会是一个噩梦，所以小程序为 live-pusher 标签支持了 debug 模式，开始 debug 模式之后，原本用于渲染视频画面的窗口上，会显示一个半透明的 log 窗口，用于展示各项音视频指标和事件，降低您调试相关功能的难度，具体使用方法我们在 FAQ 中有详细说明。

对象操作

- **wx.createLivePlayerContext()**
  通过 wx.createLivePlayerContext() 可以将 <live-player> 标签和 javascript 对象关联起来，之后即可操作该对象。

- **play**
  开始播放，如果 <live-player> 的 autoplay 属性设置为 false（默认值），那么就可以使用 play 来手动启动播放。

- **stop**
  停止播放。

- **pause**
  暂停播放，停留在最后画面

- **resume**
  继续播放，与pause成对使用

- **mute**
  设置静音。

- **requestFullScreen**
  进入全屏幕。

- **exitFullScreen**
  退出全屏幕。

```
var player = wx.createLivePlayerContext('pusher');
player.requestFullScreen({
success: function(){
console.log('enter full screen mode success!')
}
fail: function(){
console.log('enter full screen mode failed!')
}
complete: function(){
console.log('enter full screen mode complete!')
}
});
```

内部事件

通过 **<live-player>** 标签的 **bindstatechange** 属性可以绑定一个事件处理函数，该函数可以监听推流模块的内部事件和异常通知。

1. 关键事件

code	事件定义	含义说明
2001	PLAY_EVT_CONNECT_SUCC	已经连接到云端服务器
2002	PLAY_EVT_RTMP_STREAM_BEGIN	服务器开始传输音视频数据

code	事件定义	含义说明
2003	PLAY_EVT_RCV_FIRST_I_FRAME	网络接收到首段音视频数据
2004	PLAY_EVT_PLAY_BEGIN	视频播放开始，可以在收到此事件之前先用默认图片代表等待状态
2006	PLAY_EVT_PLAY_END	视频播放结束
2007	PLAY_EVT_PLAY_LOADING	进入缓冲中状态，此时播放器在等待或积攒来自服务器的数据
-2301	PLAY_ERR_NET_DISCONNECT	网络连接断开，且重新连接亦不能恢复，播放器已停止播放

播放 HTTP:// 打头的 FLV 协议地址时，如果观众遇到播放中直播流断开的情况，小程序是不会抛出 PLAY_EVT_PLAY_END 事件的，这是因为 FLV 协议中没有定义停止事件，所以只能通过监听 PLAY_ERR_NET_DISCONNECT 来替代之。

2. 警告事件

内部警告并非不可恢复的错误，小程序内部的音视频 SDK 会启动相应的恢复措施，警告的目的主要用于提示开发者或者最终用户，比如：

code	事件定义	含义说明
2101	PLAY_WARNING_VIDEO_DECODE_FAIL	当前视频帧解码失败
2102	PLAY_WARNING_AUDIO_DECODE_FAIL	当前音频帧解码失败
2103	PLAY_WARNING_RECONNECT	网络断连, 已启动自动重连恢复 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
2104	PLAY_WARNING_RECV_DATA_LAG	视频流不太稳定，可能是观看者当前网速不充裕
2105	PLAY_WARNING_VIDEO_PLAY_LAG	当前视频播放出现卡顿
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解
2107	PLAY_WARNING_VIDEO_DISCONTINUITY	当前视频帧不连续，视频源可能有丢帧，可能会导致画面花屏
3001	PLAY_WARNING_DNS_FAIL	DNS解析失败（仅播放 RTMP:// 地址时会抛送）

code	事件定义	含义说明
3002	PLAY_WARNING_SEVER_CONN_FAIL	服务器连接失败（仅播放 RTMP:// 地址时会抛送）
3003	PLAY_WARNING_SHAKE_FAIL	服务器握手失败（仅播放 RTMP:// 地址时会抛送）

3. 示例代码

```
Page({
onPlay: function(ret) {
if(ret.detail.code == 2004) {
console.log('视频播放开始',ret);
}
},

/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
//...
}
})
```

特别说明

1. <live-player> 组件是由客户端创建的原生组件，它的层级是最高的，可以使用 <cover-view> 和 <cover-image> 覆盖在上面。

2. 请勿在 <scroll-view> 中使用 <live-player> 组件。

rtc-room 标签

最近更新时间：2018-07-11 11:40:40

标签说明

<rtc-room> 标签是基于 <live-pusher> 和 <live-player> 实现的用于双人和多人音视频通话的自定义组件，其主要用于多对多音视频通话场景下（有别于<live-room>标签）。

效果演示

您可以扫描如下二维码，或在微信小程序里搜索“腾讯视频云”，即可打开我们的 demo 小程序，内部的 **双人音视频** 和 **多人音视频** 即为 <rtc-room> 的典型应用场景。

标签详解

属性定义

属性	类型	值	说明
template	String	'1v3','1v1'	必要，标识组件使用的界面模版（用户如果需要自定义界面，请看界面定制）
roomID	String	''	可选，房间号
roomName	String	''	可选，房间名
beauty	Number	0~5	可选，默认5, 美颜级别 0～5
muted	Boolean	true, false	可选，默认false，是否静音
debug	Boolean	true, false	可选，默认false，是否打印推流debug信息
bindonRoomEvent	function		必要，监听<rtc-room>组件返回的事件

操作接口

<rtc-room> 组件包含如下操作接口，您需要先通过 selectComponent 获取 <rtc-room> 标签的引用，之后就可以进行相应的操作了。

函数名	说明
start()	启动
pause()	暂停
resume()	恢复
stop()	停止
switchCamera()	切换摄像头
sendTextMsg(text:String)	发送文本消息

```
var rtcroom = this.selectComponent("#rtcroomid")
rtcroom.pause();
```

事件通知

<rtc-room> 标签通过 **onRoomEvent** 返回内部事件，事件参数格式如下

```
"detail": {
"tag": "事件tag标识，具有唯一性",
```

```
"code": "事件代码",
"detail": "对应事件的详细参数"
}
```

示例代码

```
// Page.wxml 文件
<rtc-room id="rtcroomid"
roomID="{{roomID}}"
roomName="{{roomName}}"
template="1v3"
beauty="{{beauty}}"
muted="{{muted}}"
debug="{{debug}}"
bindonRoomEvent="onRoomEvent">
</rtc-room>


// Page.js 文件
Page({
data: {
//...
roomID: '',
roomName: '',
beauty: 3,
muted: false,
debug: false,
},
//...
onRoomEvent: function(e){
switch(e.detail.tag){
case 'roomClosed': {
//房间已经关闭
break;
}
case 'error': {
//发生错误
var code = e.detail.code;
var detail = e.detail.detail;
break;
}
case 'recvTextMsg': {
//收到文本消息
var text = e.detail.detail;
```

```
break;
}
}
},

onShow: function () {
},

onHide: function () {
},

onRead: function() {
var rtcroom = this.selectComponent("#rtcroomid")
this.setData({
roomName: '测试',
});
rtcroom.start();
},

})
```

使用指引

step1: 开通相关云服务

小程序音视频依赖腾讯云提供的直播（LVB）和云通讯（IM）两项基础服务，您可以点击链接免费开通，其中云通讯服务开通即可使用，直播服务由于涉黄涉政风险较大，需要腾讯云人工审核开通。

step2: 下载自定义组件源码

<rtc-room> 并非微信小程序原生提供的标签，而是一个自定义组件，所以您需要额外的代码来支持这个标签。点击 小程序源码 下载源码包，您可以在 `wxlite` 文件夹下获取到所需文件。

step3: 登录房间服务（必需）

在使用 **<rtc-room>** 标签前需要先调用 `/utils/rtcroom.js` 的 `login` 方法进行登录，登录的目的是要连接后台房间服务（RoomService）。

```
var rtcroom = require('/utils/rtcroom.js');
...
rtcroom.login({
serverDomain: '',
userID: '',
userSig: '',
```

```
sdkAppID: '',
accType: '',
userName: '' //用户昵称，由客户自定义
});
```

参考 DOC 可以了解上面的这些参数应该怎么填写。

step4: 获取房间列表（可选）

如果您不想自己实现房间列表，而是使用房间服务自带的房间列表，您可以通过调用 `/utils/rtcroom.js` 的 `getRoomList` 函数获取到列表信息。

```
var rtcroom = require('/utils/rtcroom.js');
...
rtcroom.getRoomList({
data: {
index: 0, //列表索引号
cnt: 20 //要拉取的列表个数
},
success: function (ret) {
console.log('获取房间列表:', ret.rooms);
},
fail: function (ret) {
console.error('获取房间列表失败:', ret);
}
});
```

step5: 在工程中引入组件

- 在 page 目录下的 json 配置文件内引用组件，这一步是必须的，因为 <rtc-room> 并非原生标签。

```
"usingComponents": {
"rtc-room": "/pages/rtc-room/rtc-room"
}
```

- 在 page 目录下的 wxml 文件中使用标签

```
<rtc-room id="rtcroomid"
roomID="{{roomID}}"
roomName="{{roomName}}"
template="1v3"
beauty="{{beauty}}"
muted="{{muted}}"
```

```
debug="{{debug}}"
bindonRoomEvent="onRoomEvent">
</rtc-room>
```

step6: 如何创建房间

- 如果您希望由后台房间服务（RoomService）自动分配一个 roomid， 那么您只需要给 <rtc-room> 指定 roomName 就可以。

```
//创建房间（RoomService 自动分配 roomid）
this.setData({
roomName: '测试'
});
rtcroom.start();
```

- 如果您希望自己指定 roomid， 那么您需要先设定 roomID 属性，才可以调用 start() 函数。

```
//创建房间 (由您来指定 roomid)
this.setData({
roomID: 12345
});
rtcroom.start();
```

- 不论哪种方案，只有 **start()** 函数被调用时，房间才会真正的被创建。

step7: 如何加入房间

- 如果一个 roomid 对应的房间已经被创建了，那么 start() 就不再是创建房间，而是直接进入房间。

```
this.setData({
roomID: 12345
});
rtcroom.start();
```

界面定制

如果我们默认实现的 "1v1" 和 "1v3" 两种界面布局不符合您的要求，您也可以根据项目需要对界面进行定制：

- **创建界面模版**

```
//第一步：新建 /pages/templates/mytemplate 文件夹，并创建 mytemplate.wxml 和 mytemplate.wxss 文件

//第二步：编写 mytemplate.wxml 和 mytemplate.wxss 文件
//mytemplate.wxml
<template name='mytemplate'>
<view class='videoview'>
<view class="pusher-box">
<live-pusher
id="rtcpusher"
autopush
mode="RTC"
url="{{pushURL}}"
aspect="{{aspect}}"
min-bitrate="{{minBitrate}}"
max-bitrate="{{maxBitrate}}"
audio-quality="high"
beauty="{{beauty}}"
muted="{{muted}}"
waiting-image="https://mc.qcloudimg.com/static/img/
daeed8616ac5df256c0591c22a65c4d3/pause_publish.jpg"
background-mute="{{true}}"
debug="{{debug}}"
bindstatechange="onPush"
binderror="onError">
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>我</cover-view>
</live-pusher>
</view>
<view class="player-box" wx:for="{{members}}" wx:key="userID">
<view class='poster'>
<cover-image class='set'
src="https://miniprogram-1252463788.file.myqcloud.com/roomset_{{index + 2}}.png">
</cover-image>
</view>
<live-player
id="{{item.userID}}"
autoplay
mode="RTC"
wx:if="{{item.accelerateURL}}"
object-fit="fillCrop"
min-cache="0.1"
max-cache="0.3"
src="{{item.accelerateURL}}"
debug="{{debug}}"
```

```
background-mute="{{true}}"
bindstatechange="onPlay">
<cover-view class='loading' wx:if="{{item.loading}}">
<cover-image src="/pages/Resources/loading_image0.png"></cover-image>
</cover-view>
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>{{item.userName}}</cover-view>
</live-player>
</view>
</view>
</template>

//mytemplate.wxss
.videoview {
background-repeat:no-repeat;
background-size: cover;
width: 100%;
height: 100%;
}
```

- **rtc-room 组件引入模版**

```
//为 <rtc-room> 组件中的 liveroom.wxml 文件添加自定义模版
<import src='/pages/templates/mytemplate/mytemplate.wxml'/>
<view class='conponent-box'>
<view styles="width:100%;height=100%;" wx:if="{{template=='1v3' || template=='1v1'}}">
<template is='mytemplate' data="{{pushURL, aspect,
minBitrate, maxBitrate, beauty, muted, debug, members}}"/>
</view>
</view>

//为 <rtc-room> 组件中的 liveroom.wxss 文件添加自定义样式
@import "../templates/mytemplate/mytemplate.wxss";
```

其它平台

<rtc-room> 也有 Windows、iOS、Android 等平台下的对等实现，您可以参考下表中的资料。同时， 阅读 设计文档，您可以了解该解决方案的内部设计原理。

所属平台	SDK下载	文档指引
Windows(C++)	DOWNLOAD	API

所属平台	SDK下载	文档指引
Windows(C#)	DOWNLOAD	API
IE浏览器	DOWNLOAD	API
iOS	DOWNLOAD	API
Android	DOWNLOAD	API

录制指引

- step1： 开通 腾讯云点播服务。
- step2：进入直播控制台（小程序音视频流媒体是基于直播服务构建的），在【接入管理>>接入配置>>直播录制】中，开启录制功能。（注意：这里说的录制费用是按并发收费的，不是每一路都收费）

- step3：在点播的视频管理界面中，您可以看到这些录制的文件，您也可以通过点播服务的 REST API 获取到这些文件。

live-room 标签

最近更新时间：2019-03-13 14:39:42

标签说明

<live-room> 标签是基于 <live-pusher> 和 <live-player> 实现的用于双人和多人音视频通话的自定义组件，其主要用于一对多音视频通话场景下（有别于<rtc-room>）。

效果演示

您可以扫描如下二维码，或在微信小程序里搜索“腾讯视频云”，即可打开我们的 demo 小程序，内部的 **直播体验室** 即为 <live-room> 的典型应用场景。

标签详解

属性定义

属性	类型	值	说明
template	String	'1v1' 或 '1v3'	必要，标识组件使用的界面模版（用户如果需要自定义界面，请看界面定制）
roomID	String	您来指定	可选，房间号（role = audience 时，roomID 不能为空）
roomName	String	您来指定	可选，房间名
role	String	‘presenter’, 'audience'	必要，presenter 代表主播，audience 代表观众
pureaudio	Boolean	true，false	可选，默认false，是否开启纯音频推流
beauty	Number	0~5	可选，默认5, 美颜级别 0～5
muted	Boolean	true, false	可选，默认false，是否静音
debug	Boolean	true, false	可选，默认false，是否打印推流debug信息
bindonRoomEvent	function		必要，监听rtcroom组件返回的事件

操作接口

<live-room> 组件包含如下操作接口，您需要先通过 selectComponent 获取 <live-room> 标签的引用，之后就可以进行相应的操作了。

函数名	说明
start()	启动
pause()	暂停
resume()	恢复
stop()	停止
requestJoinPusher()	请求连麦，适用于audience
respondJoinReq(agree:Boolean, audience:Object)	同意连麦，适用于presenter
switchCamera()	切换摄像头
sendTextMsg(text:String)	发送文本消息

```
var liveroom = this.selectComponent("#liveroomid")
liveroom.pause();
```

事件通知

<live-room> 标签通过 **onRoomEvent** 返回内部事件，事件参数格式如下

```
"detail": {
"tag": "事件tag标识，具有唯一性",
"code": "事件代码",
"detail": "对应事件的详细参数"
}
```

示例代码

```
// Page.wxml 文件
<live-room id="liveroomid"
roomID="{{roomID}}"
roomName="{{roomName}}"
template="1v3"
beauty="{{beauty}}"
muted="{{muted}}"
debug="{{debug}}"
bindonRoomEvent="onRoomEvent">
</live-room>


// Page.js 文件
Page({
data: {
//...
roomID: '',
roomName: '',
beauty: 3,
muted: false,
debug: false,
},
//...
onRoomEvent: function(e){
switch(e.detail.tag){
case 'roomClosed': {
//房间已经关闭
break;
```

```
}
case 'error': {
//发生错误
var code = e.detail.code;
var detail = e.detail.detail;
break;
}
case 'recvTextMsg': {
//收到文本消息
var text = e.detail.detail;
break;
}
case 'joinPusher': {
//收到来自观众的连麦请求
var audience = e.detail;
var name = audience.userName;
var id = audience.userID;
// 允许请求
liveroom.respondJoinReq(true, audience)
break;
}
}
},

onShow: function () {
},

onHide: function () {
},

onRead: function() {
var liveroom = this.selectComponent("#liveroomid");
this.setData({
roomName: '测试',
});
liveroom.start();
},

})
```

使用指引

step1: 开通相关云服务

小程序音视频依赖腾讯云提供的直播（LVB）和云通信（IM）两项基础服务，您可以单击链接开通，其中云通讯服务开通即可使用，直播服务由于涉黄涉政风险较大，需要腾讯云人工审核开通。

step2: 下载自定义组件源码

<live-room> 并非微信小程序原生提供的标签，而是一个自定义组件，所以您需要额外的代码来支持这个标签。单击 小程序源码 下载源码包，您可以在 `wxlite` 文件夹下获取到所需文件。

step3: 登录房间服务（必需）

在使用 **<live-room>** 标签前需要先调用 `/utils/liveroom.js` 的 `login` 方法进行登录，登录的目的是要连接后台房间服务（RoomService）。

```
var liveroom = require('/utils/liveroom.js');
...
liveroom.login({
serverDomain: '',
userID: '',
userSig: '',
sdkAppID: '',
accType: '',
userName: '' //用户昵称，由客户自定义
});
```

参考 直播连麦（LiveRoom） 可以了解上面的这些参数应该怎么填写。

step4: 获取房间列表（可选）

如果您不想自己实现房间列表，而是使用房间服务自带的房间列表，您可以通过调用 `/utils/liveroom.js` 的 `getRoomList` 函数获取到列表信息。

```
var liveroom = require('/utils/liveroom.js');
...
liveroom.getRoomList({
data: {
index: 0, //列表索引号
cnt: 20 //要拉取的列表个数
},
success: function (ret) {
console.log('获取房间列表:', ret.rooms);
},
fail: function (ret) {
console.error('获取房间列表失败:', ret);
}
});
```

step5: 在工程中引入组件

- 在 page 目录下的 json 配置文件内引用组件，这一步是必须的，因为 <live-room> 并非原生标签。

```
"usingComponents": {
"live-room": "/pages/liveroom_component/liveroom"
}
```

step6: 主播：创建房间

- 在 page 目录下的 wxml 文件中使用标签 <live-room>，并将 role 指定为 **presenter**。

```
<live-room id="liveroomid"
template="1v3"
role="presenter"
roomID="{{roomID}}"
roomName="测试房间"
pureaudio="false",
beauty="5",
debug="true" >
</live-room>
```

- 如果您希望由后台房间服务（RoomService）自动分配一个 roomID， 那么您只需要给 <live-room> 指定 roomName 就可以。

```
//创建房间（RoomService 自动分配 roomID）
this.setData({
roomName: '测试'
});
var liveroom = this.selectComponent("#liveroomid");
liveroom.start();
```

- 如果您希望自己指定 roomID， 那么您需要先设定 roomID 属性，才可以调用 start() 函数。

```
//创建房间 (由您来指定 roomID)
this.setData({
roomID: 12345
});
var liveroom = this.selectComponent("#liveroomid");
liveroom.start();
```

- 不论哪种方案，只有 **start()** 函数被调用时，房间才会真正的被创建。

step7: 观众：加入房间

- 在 page 目录下的 wxml 文件中使用标签 <live-room>，并将 role 指定为 **audience**。

```
<live-room id="liveroomid"
template="1v3"
role="audience"
roomID="{{roomID}}"
pureaudio="false",
beauty="5",
debug="true" >
</live-room>
```

- 如果一个 roomID 对应的房间已经被创建了，那么 start() 就不再是创建房间，而是直接进入房间。

```
//创建房间 (由您来指定 roomID)
this.setData({
roomID: 12345
});
var liveroom = this.selectComponent("#liveroomid");
liveroom.start();
```

step8: 连麦互动

- 观众：可以向主播发起连麦请求

```
var liveroom = this.selectComponent("#liveroomid");
liveroom.requestJoinPusher();
```

- 主播：可以接受或者拒绝连麦请求

```
var liveroom = this.selectComponent("#liveroomid");
liveroom.respondJoinReq(true, aduience);
```

界面定制

如果我们默认实现的 "1v1" 和 "1v3" 两种界面布局不符合您的要求，您也可以根据项目需要对界面进行定制：

- **创建界面模版**

```
//第一步：新建 /pages/templates/mytemplate 文件夹，并创建 mytemplate.wxml 和 mytemplate.wxss 文件

//第二步：编写 mytemplate.wxml 和 mytemplate.wxss 文件
//mytemplate.wxml
<template name='mytemplate'>
<view class='inner-container'>
<live-pusher wx:if="{{isCaster&&mainPusherInfo.url}}" id="pusher" mode="RTC" enable-camera="{{true}}" url="{{mainPusherInfo.url}}" beauty="{{beauty}}" muted="{{muted}}" aspect="{{mainPusherInfo.aspect}}" waiting-image="https://mc.qcloudimg.com/static/img/daeed8616ac5df256c0591c22a65c4d3/pause_publish.jpg"
background-mute="{{true}}" debug="{{debug}}" bindstatechange="onMainPush" binderror="onMainError">
<!-- <cover-image class='character' src="/pages/Resources/mask.png"></cover-image> -->
<cover-view class='character' style='padding: 0 5px;'>我 ( {{userName}} ) </cover-view>
</live-pusher>

<block wx:for="{{visualPlayers}}" wx:key="{{index}}">
<live-player wx:if="{{item.url}}" autoplay id="player" mode="{{item.mode}}" object-fit="fillCrop" src="{{item.url}}" debug="{{item.debug}}" background-mute="{{item.mute}}" bindstatechange="onMainPlayState" binderror="onMainPlayError">
<cover-view class='loading' wx:if="{{item.loading}}">
<cover-image src="/pages/Resources/loading_image0.png"></cover-image>
</cover-view>
<!-- <cover-image class='character' src="/pages/Resources/mask.png"></cover-image> -->
<cover-view class='character' style='padding: 0 5px;'>{{item.userName}}</cover-view>
</live-player>
</block>
</view>

<view class='list-container'>
<view class='.list-item-box' wx:if="{{!isCaster && linkPusherInfo.url}}">
<live-pusher wx:if="{{!isCaster && linkPusherInfo.url}}" id="audience_pusher" mode="RTC" url="{{linkPusherInfo.url}}" beauty="{{beauty}}" muted="{{muted}}"
aspect="{{linkPusherInfo.aspect ? linkPusherInfo.aspect : '3:4'}}" waiting-image="https://mc.qcloudimg.com/static/img/daeed8616ac5df256c0591c22a65c4d3/pause_publish.jpg"
background-mute="true" debug="{{debug}}" bindstatechange="onLinkPush" binderror="onLinkError">
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>我 ( {{userName}} ) </cover-view>
<cover-view class='close-ico' bindtap="quitLink">x</cover-view>
```

```
</live-pusher>
</view>

<view class='.list-item-box' wx:for="{{members}}" wx:key="{{item.userID}}">
<live-player id="{{item.userID}}" autoplay mode="RTC" object-fit="fillCrop" min-cache="0.1" max-ca
che="0.3" src="{{item.accelerateURL}}" debug="{{debug}}" background-mute="{{true}}">
<cover-view class="close-ico" wx:if="{{item.userID == userID || isCaster}}" bindtap="kickoutSubPushe
r" data-userid="{{item.userID}}">x</cover-view>
<cover-view class='loading' wx:if="{{false}}">
<cover-image src="/pages/Resources/loading_image0.png"></cover-image>
</cover-view>
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>{{item.userName}}</cover-view>
</live-player>
</view>
</view>
</template>

//mytemplate.wxss
.videoview {
background-repeat:no-repeat;
background-size: cover;
width: 100%;
height: 100%;
}
```

- **live-room 组件引入模版**

```
//为 <live-room> 组件中的 liveroom.wxml 文件添加自定义模版
<import src='/pages/templates/mytemplate/mytemplate.wxml'/>
<view class='conponent-box'>
<view styles="width:100%;height=100%;" wx:if="{{template=='1v3' || template=='1v1'}}">
<template is='mytemplate' data="{{pushURL, aspect,
minBitrate, maxBitrate, beauty, muted, debug, members}}"/>
</view>
</view>

//为 <live-room> 组件中的 liveroom.wxss 文件添加自定义样式
@import "../templates/mytemplate/mytemplate.wxss";
```

其它平台

<live-room> 也有 Windows、iOS、Android 等平台下的对等实现，您可以参考下表中的资料。同时， 阅读 视频通话（ RTCRoom ），您可以了解该解决方案的内部设计原理。

所属平台	SDK下载	文档指引
Windows(C++)	DOWNLOAD	API
Windows(C#)	DOWNLOAD	API
IE浏览器	DOWNLOAD	API
iOS	DOWNLOAD	API
Android	DOWNLOAD	API

录制指引

- step1： 开通 腾讯云点播服务。
- step2：进入直播控制台（小程序音视频流媒体是基于直播服务构建的），在【接入管理>>接入配置>>直播录制】中，开启录制功能。（注意：这里说的录制费用是按并发收费的，不是每一路都收费）

- step3：在点播的视频管理界面中，您可以看到这些录制的文件，您也可以通过点播服务的 REST API 获取到这些文件。

webrtc-room

最近更新时间：2018-10-18 14:57:32

标签说明

<webrtc-room> 标签是基于 <live-pusher> 和 <live-player> 实现的用于 WebRTC 互通的自定义组件。如果您希望直接使用 <live-pusher> 和 <live-player> 标签完成对接，或者想要了解 <webrtc-room> 的内部原理，可以参考 DOC。

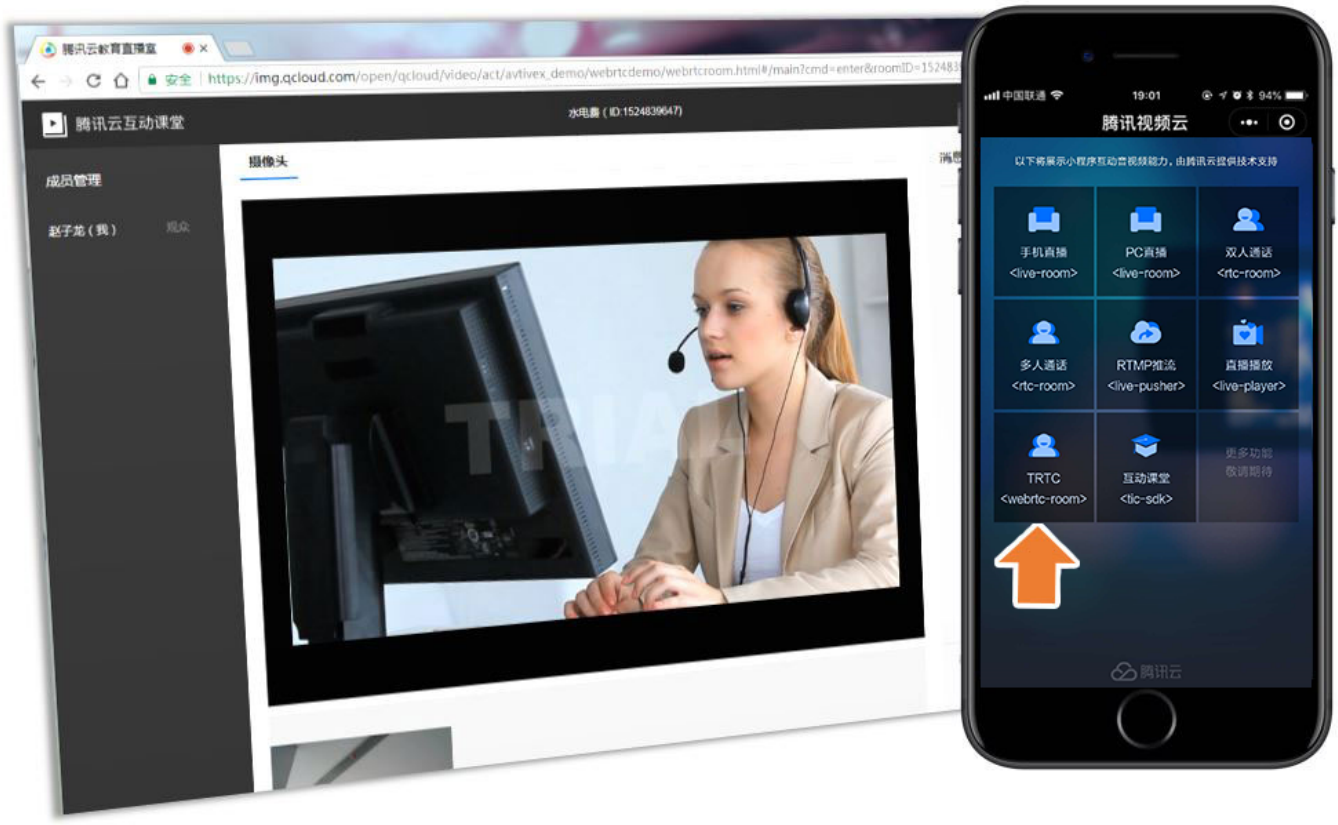
版本要求

- 微信 6.6.6 版本开始支持。

效果演示

- **PC 端**
  用 Chrome 浏览器打开 体验页面 可以体验桌面版 WebRTC 的效果。

- **微信端**
  发现=>小程序=>搜索“腾讯视频云”，点击 WebRTC 功能卡，就可以体验跟桌面版 Chrome 互通的效果了。



对接资料

对接资料	说明	github地址
小程序源码	包含<webrtc-room>的组件源码以及demo源码	前往
PC端源码	基于WebrtcAPI实现的Chrome版WebRTC接入源码（其中component/WebRTCRoom.js 实现了一个简单的房间管理功能，component/mainwindow.js包含了对 WebRTC API 的使用代码）	前往
后台源码	实现了一个简单的房间列表功能，同时包含<webrtc-room>几个所需参数的生成代码	前往

标签详解

属性定义

属性	类型	值	说明
template	String	'1v3'	必要，标识组件使用的界面模版（用户如果需要自定义界面，请看界面定制）
sdkAppID	String	''	必要，开通IM服务所获取到的AppID
userID	String	''	必要，用户ID
userSig	String	''	必要，身份签名，相当于登录密码的作用
roomID	Number	''	必要，房间号
privateMapKey	String	''	必要，房间权限key，相当于进入指定房间roomID的钥匙
beauty	Number	0~5	可选，默认5, 美颜级别 0～5
muted	Boolean	true, false	可选，默认false，是否静音
debug	Boolean	true, false	可选，默认false，是否打印推流debug信息
bindRoomEvent	function		必要，监听<webrtc-room>组件返回的事件
enableIM	Boolean	true, false	可选，默认false
bindIMEvent	function		当IM开启时必要，监听IM返回的事件

操作接口

<webrtc-room> 组件包含如下操作接口，您需要先通过 selectComponent 获取 <webrtc-room> 标签的引用，之后就可以进行相应的操作了。

函数名	说明
start()	启动
pause()	暂停
resume()	恢复
stop()	停止
switchCamera()	切换摄像头

```
var webrtcroom = this.selectComponent("#webrtcroomid")
webrtcroom.pause();
```

事件通知

<webrtc-room> 标签通过 **onRoomEvent** 返回内部事件，通过 **onIMEvent** 返回IM消息事件，事件参数格式如下

```
"detail": {
"tag": "事件tag标识，具有唯一性",
"code": "事件代码",
"detail": "对应事件的详细参数"
}
```

示例代码

```
// Page.wxml 文件
<webrtc-room id="webrtcroom"
roomID="{{roomID}}"
userID="{{userID}}"
userSig="{{userSig}}"
sdkAppID="{{sdkAppID}}"
privateMapKey="{{privateMapKey}}"
template="1v3"
beauty="{{beauty}}"
muted="{{muted}}"
debug="{{debug}}"
bindRoomEvent="onRoomEvent"
enableIM="{{enableIM}}"
bindIMEvent="onIMEvent">
</webrtc-room>


// Page.js 文件
Page({
data: {
//...
roomID: '',
userID: '',
userSig: '',
sdkAppID: '',
beauty: 3,
muted: false,
```

```
debug: false,
enableIM: false
},
onRoomEvent: function(e){
switch(e.detail.tag){
case 'error': {
//发生错误
var code = e.detail.code;
var detail = e.detail.detail;
break;
}
}
},
onIMEvent: function(e){
switch(e.detail.tag){
case 'big_group_msg_notify':
//收到群组消息
console.debug( e.detail.detail )
break;
case 'login_event':
//登录事件通知
console.debug( e.detail.detail )
break;
case 'connection_event':
//连接状态事件
console.debug( e.detail.detail )
break;
case 'join_group_event':
//进群事件通知
console.debug( e.detail.detail )
break;
}
},

onLoad: function (options) {
self.setData({
userID: self.data.userID,
userSig: self.data.userSig,
sdkAppID: self.data.sdkAppID,
roomID: self.data.roomID,
privateMapKey: res.data.privateMapKey
}, function() {
var webrtcroomCom = this.selectComponent('#webrtcroom');
if (webrtcroomCom) {
webrtcroomCom.start();
```

```
}
})
},

})
```

使用指引

step1: 开通相关云服务

小程序跟 WebRTC 的互通是基于实时音视频（TRTC）服务实现的，需要开通该服务。

- 进入实时音视频管理控制台，如果服务还没有开通，点击申请开通，之后会进入腾讯云人工审核阶段，审核通过后即可开通。
- 服务开通后，进入管理控制台 创建实时音视频应用，点击【确定】按钮即可。

- 从实时音视频控制台获取 sdkAppID、accountType、privateKey ，在 step4 中会用的：

step2: 下载自定义组件源码

<webrtc-room> 并非微信小程序原生提供的标签，而是一个自定义组件，所以您需要额外的代码来支持这个标签。点击 小程序源码 下载源码包，您可以在 wxlite 文件夹下获取到所需文件。

step3: 在工程中引入组件

- 在 page 目录下的 json 配置文件内引用组件，这一步是必须的，因为 <webrtc-room> 并非原生标签。

```
"usingComponents": {
"webrtc-room": "/pages/webrtc-room/webrtc-room"
}
```

- 在 page 目录下的 wxml 文件中使用标签

```
<webrtc-room id="webrtcroomid"
roomID="{{roomID}}"
userID="{{userID}}"
userSig="{{userSig}}"
sdkAppID="{{sdkAppID}}"
privateMapKey="{{privateMapKey}}"
template="1v3"
beauty="{{beauty}}"
muted="{{muted}}"
debug="{{debug}}"
bindRoomEvent="onRoomEvent"
enableIM="{{enableIM}}"
bindIMEvent="onIMEvent">
</webrtc-room>
```

step4: 获取key信息

按照如下表格获取关键的key信息，这是使用腾讯云互通直播服务所必须的几个信息：

KEY	示例	作用	获取方案
sdkAppID	1400087915	用于计费和业务区分	step1 中获取
userID	xiaoming	用户名	可以由您的服务器指定，或者使用小程序的openid
userSig	加密字符串	相当于 userid 对应的登录密码	由您的服务器签发（PHP / JAVA）
roomID	12345	房间号	可以由您的服务器指定
privateMapKey	加密字符串	进房票据：相当于是进入 roomid 的钥匙	由您的服务器签发（PHP / JAVA）

下载 sign_src.zip 可以获得服务端签发 userSig 和 privateMapKey 的计算代码（生成 userSig 和 privateMapKey 的签名算法是 **ECDSA-SHA256**）。

step5: 进入房间

```
self.setData({
userID: userID,
userSig: userSig,
sdkAppID: sdkAppID,
roomID: roomID,
privateMapKey: privateMapKey
}, function() {
var webrtcroomCom = this.selectComponent('#webrtcroomid');
if (webrtcroomCom) {
webrtcroomCom.start();
}
})
```

界面定制

- **创建界面模版**

```
//第一步：新建 /pages/templates/mytemplate 文件夹，并创建 mytemplate.wxml 和 mytemplate.wxss 文件

//第二步：编写 mytemplate.wxml 和 mytemplate.wxss 文件
//mytemplate.wxml
<template name='mytemplate'>
<view class='videoview'>
<view class="pusher-box">
<live-pusher
id="rtcpusher"
autopush
mode="RTC"
url="{{pushURL}}"
aspect="{{aspect}}"
min-bitrate="{{minBitrate}}"
max-bitrate="{{maxBitrate}}"
audio-quality="high"
beauty="{{beauty}}"
muted="{{muted}}"
waiting-image="https://mc.qcloudimg.com/static/img/
daeed8616ac5df256c0591c22a65c4d3/pause_publish.jpg"
background-mute="{{true}}"
debug="{{debug}}"
bindstatechange="onPush"
binderror="onError">
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>我</cover-view>
</live-pusher>
</view>
<view class="player-box" wx:for="{{members}}" wx:key="userID">
<view class='poster'>
<cover-image class='set'
src="https://miniprogram-1252463788.file.myqcloud.com/roomset_{{index + 2}}.png">
</cover-image>
</view>
<live-player
id="{{item.userID}}"
autoplay
mode="RTC"
wx:if="{{item.accelerateURL}}"
object-fit="fillCrop"
min-cache="0.1"
max-cache="0.3"
src="{{item.accelerateURL}}"
debug="{{debug}}"
```

```
background-mute="{{true}}"
bindstatechange="onPlay">
<cover-view class='loading' wx:if="{{item.loading}}">
<cover-image src="/pages/Resources/loading_image0.png"></cover-image>
</cover-view>
<cover-image class='character' src="/pages/Resources/mask.png"></cover-image>
<cover-view class='character' style='padding: 0 5px;'>{{item.userName}}</cover-view>
</live-player>
</view>
</view>
</template>

//mytemplate.wxss
.videoview {
background-repeat:no-repeat;
background-size: cover;
width: 100%;
height: 100%;
}
```

- **webrtc-room 组件引入模版**

```
//为 <webrtc-room> 组件中的 webrtcroom.wxml 文件添加自定义模版
<import src='/pages/templates/mytemplate/mytemplate.wxml'/>
<view class='conponent-box'>
<view styles="width:100%;height=100%;" wx:if="{{template=='1v3'}}">
<template is='mytemplate' data="{{pushURL, aspect,
minBitrate, maxBitrate, beauty, muted, debug, members}}"/>
</view>
</view>

//为 <webrtc-room> 组件中的 webrtcroom.wxss 文件添加自定义样式
@import "../templates/mytemplate/mytemplate.wxss";
```

Chrome端对接

了解腾讯云官网的 WebrtcAPI ，可以对接 Chrome 端的 H5 视频通话，因为不是本文档的重点，此处不做赘述。

企业端
WebEXE

最近更新时间：2018-09-21 19:20:41

方案选型

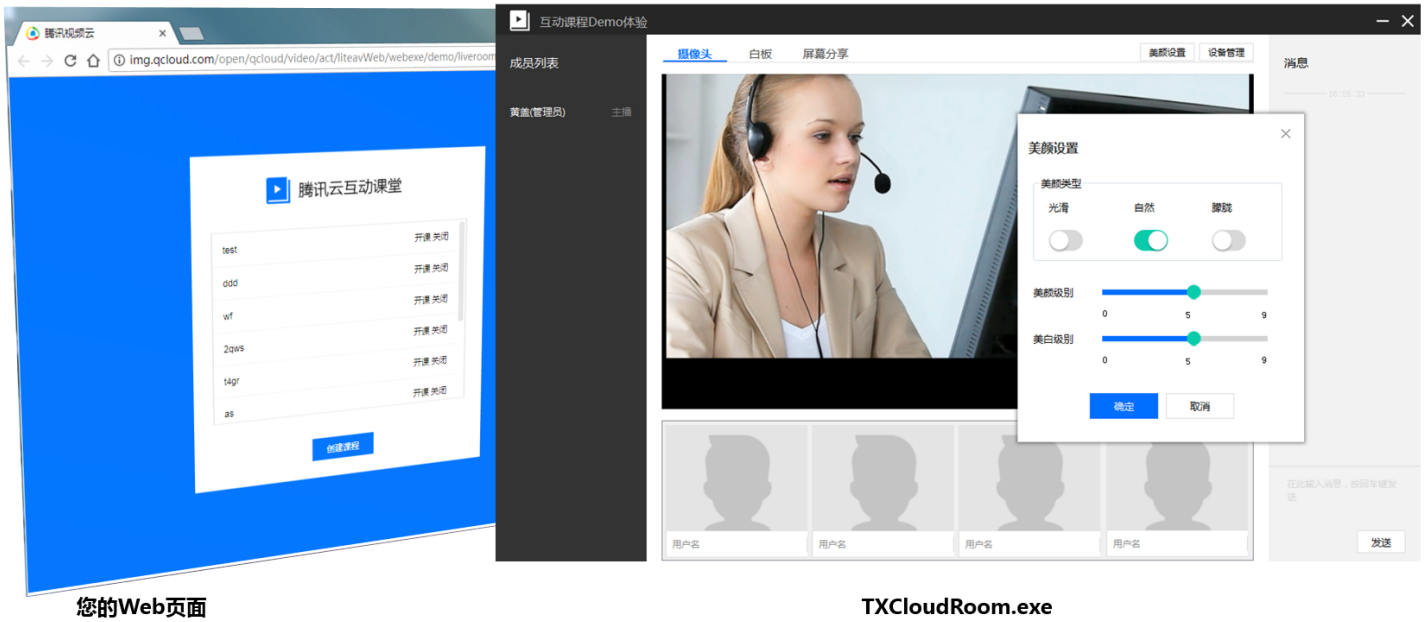
WebEXE 和 WebRTC 是我们推出的两套企业端接入方案，下表列出了两套方案的适用场景和优缺点，您可以根据自己的情况自行选择。

方案选型	WebEXE	WebRTC
文档地址	DOC	DOC
适用场景	面向公司职员	面向普通C类用户
方案优势	可以跳开浏览器的各种限制，实现一些高级特性	无需安装插件，Chrome浏览器就能胜任，适合普通用户接入
方案不足	需要使用者按提示安装程序	功能受到Chrome浏览器的安全限制
美颜磨皮	支持美颜	不支持美颜
桌面录屏	支持桌面录屏	需要安装录屏插件
本地录制	支持本地录制	不支持本地录制
依赖的云服务	LVB + IM	TRTC + IM

Demo体验

用任意浏览器打开 体验地址 即可了解 WebEXE 方案，左侧的网页可以替换成您的Web页面，右侧的 TXCloudRoom.exe 用于实现视频通话等功能。

- **网页（Web）**：承载您原有的业务系统和业务逻辑，比如订单系统，CRM系统等各种电子流系统。
- **桌面程序（EXE）**：类似PC版微信这样的应用程序，能够被您的网页直接唤起。具有性能优异，稳定性好等特点，能实现一些浏览器能力范围之外的功能。



源码调试

PC 网页

点击 GitHub 下载网页端源代码，用本地浏览器双击打开文件中的 index.html，就可以体验和调试 WebEXE 的相关功能。首次使用时，会提示需要下载和安装本地客户端插件。

目录	说明
index.html	demo主页面
doubleroom.html	双人视频通话的demo页面
liveroom.html	互动视频通话的demo页面
assets	demo页面中使用的 css 样式表和资源文件
js	demo页面中使用的javascript，其中，最为关键的 EXEStart.js就在这里
exe	包含TXCloudRoomSetup.exe安装包

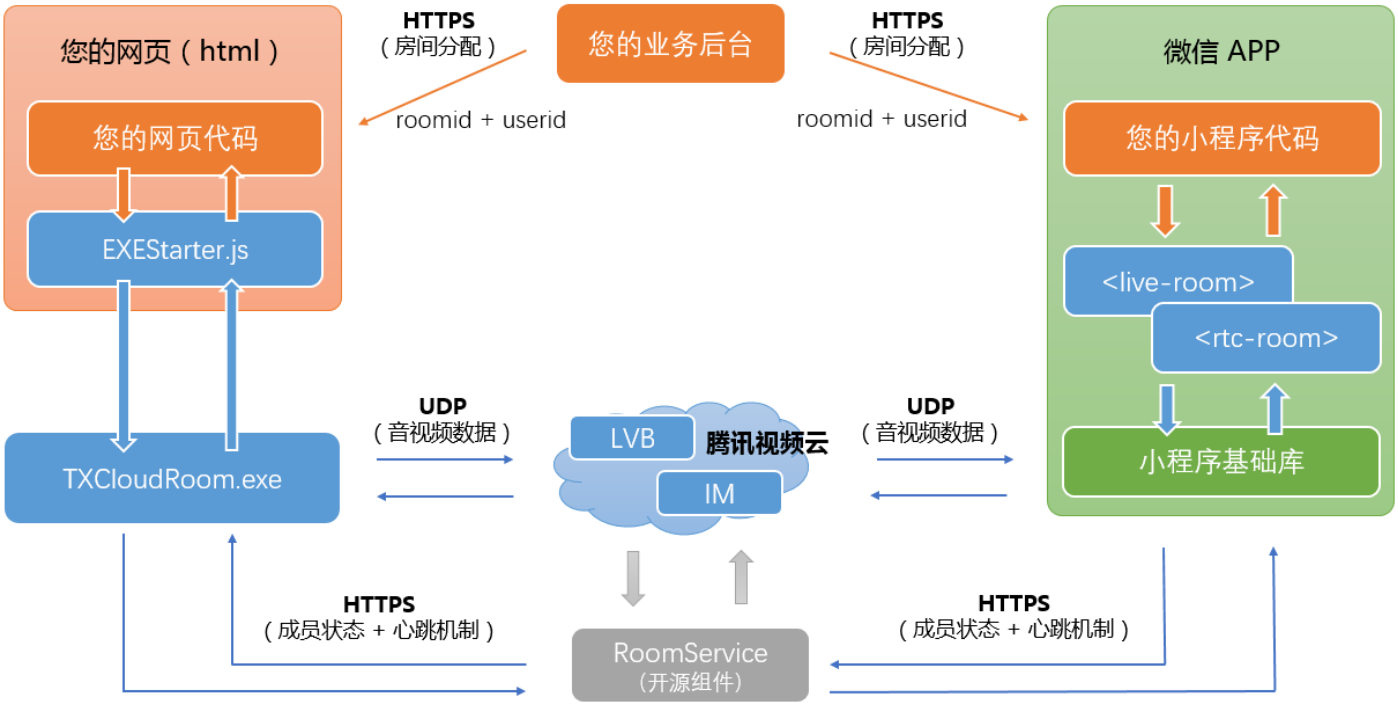
Server

点击 GitHub 可以下载一份 **java** 版本的 Server 端源码，这份代码的主要作用是实现了一个简单的（无鉴权的）房间列表，可以支持创建通话房间、关闭通话房间等功能。如果您只是希望打通视频通话（在 PC 网页和小程序各写死一个 roomid），则不太需要这部分代码的帮助。

目录	说明
README.pdf	介绍了如何使用这份后台代码
后台接口表.pdf	介绍了这份后台代码的内部实现细节
src	java 版本的后台房间列表源代码

方案对接

下面这幅图简单介绍了如何将 WebEXE 方案整合到您的现有的业务系统中：



step1: 搭建业务服务器

业务服务器的作用主要是向PC端网页和微信小程序派发 roomid、userid、usersig 这些进行视频通话所必须的信息。其中roomid 和 userid 都可以由您的业务后台自由决定，只要确保不会出现 id重叠 就可以。usersig 的计算则需要参考 DOC，我们在官网也提供了 java 和 php 版本的计算源码。

step2: 部署RoomService

WebEXE 实现视频通话服务所使用的 LiveRoom(直播+连麦) 和 RTCRoom(视频通话) 组件，都依赖一个叫做 RoomService 的后台开源组件（JAVA | Node.js）用于实现视频房间逻辑，您可以点击 RoomService.zip 下载到相关代码，部署方法见 zip 包中的说明文档 **README.pdf**。

step3: 对接PC Web端代码

您的web页面需要 include **EXEStarter.js** 这个 javascript 文件，并且把 step1 中获取的 roomid, userid, usersig 这些信息都传递给 **EXEStarter.js** 的 **startEXE** 函数。其中几个关键参数这里详细说明一下：

参数	详细说明
sdkAppID	腾讯云通讯服务用 sdkAppID 区分 IM 客户身份，参考 DOC 了解怎么获取
accType	曾用于区分 APP 类型，现仅出于兼容性原因而保，参考 DOC 了解怎么获取
userID	用户ID，您的业务服务器负责分配，各个端不能重复，否则会出现"被踢下线"的情况
userSig	相当于用户密码，具体怎么计算，可以参考 DOC 了解
serverDomain	RoomService 地址，step2中部署完RoomService之后即可获得
roomId	房间ID，您的业务服务器负责分配，小程序端和PC端进入同一个ID的房间，即可进行视频通话
type	RTCRoom 和 LiveRoom 两种模式，其区别可以看 step4
template	视频窗口摆放样式，默认1V1，更多参考 Template 定义
mixRecord	云端录制，在这种录制模式下，EXE并不参与录制工作，所有录制均在后台进行，因此 WebEXE 和 WebRTC 两种解决方案都通用，但也存在缺乏定制型的缺点。
screenRecord	本地录制，是WebEXE特有的录制模式，EXE程序直接抓取本地的画面并实时生成录制影片，根据参数不同，EXE会将生成的影片文件存在本地或者推到云端。
cloudRecordUrl	如果 screenRecord 选择 RecordScreenToServer 或者 RecordScreenToBoth，需要指定一个 rtmp:// 推流地址，这样视频流就能按照正常的推流录制模式，直接将影片内容录制到云端。

EXEStarter.js 主要用于唤起 TXCloudRoom.exe 桌面程序，并跟 TXCloudRoom.exe 进行双向通讯，您的 Web 页面只需要处理房间管理等逻辑，音视频相关的复杂功能，则交给 TXCloudRoom.exe 去完成。

点击示例代码，可以查看一个简单的唤起 TXCloudRoom.exe 的程序，您也可以在 GitHub 上获取一份更加完善的适用于 PC 端网页的源代码。

step4: 对接小程序端代码

小程序端的对接参考微信端的文档：

文档链接	适合场景

文档链接	适合场景
<rtc-room>	纯视频通话场景，比如双人1v1视频通话，或者视频会议
<live-room>	直播+连麦混合场景，基于LVB服务实现，所以既能以很低的带宽成本支持上千人的并发观看，又能支持观众和主播之间的实时视频通话

如何录制

您可以把用户整个直播过程录制下来，然后作为视频文件用于回看。具体如何实现录制功能，可以查看全程录制。

全程录制

最近更新时间：2018-07-11 11:51:45

功能介绍

您可以把用户整个直播过程录制下来，然后作为视频文件用于回看。适用于记录在线定损、互动课堂、远程庭审等场景。

开启录制

录制回看功能依托于腾讯云的**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点播服务。服务开通之后，新录制的文件就可以在点播控制台的 视频管理 里找到它们。

那么怎么开启录制呢？这里有两个办法：

1. 全局开启录制

即指定所有直播的视频流全部开启或者关闭录制，在 直播管理控制台 >> 接入管理 >> 直播码接入 >> 接入配置 中可以对其进行设置，如下图：

这里有几点要特别说明：

- 目前全局录制功能支持的视频封装格式有 mp4、 hls 和 flv。
- 您可指定同时录制一种以上的视频格式（只有 mp4 和 hls 支持手机浏览器播放）。
- mp4 格式的视频是 不支持 直播中途切换分辨率或做横竖屏切换的。若您需要屏幕分享功能（推流过程中会涉及到屏幕分辨率变化），若没有开启混流，则推荐使用 hls 封装格式。
- 如果您指定的录制格式是 flv 或 mp4 ，全局录制的分片时长默认为30分钟。如果您需要配置全局录制的分片时长，可以提工单申请。
- hls（m3u8） 文件本身就是小分片机制，所以无所谓录制时长，只要直播过程中不断流，您只会拿到一个m3u8文件。但如果直播期间推流出现中断，录制过程依然会出现分段问题（也就是会得到多个 m3u8），其中一个常见的问题就是 App 切后台，推荐采用后台推流解决方案进行缓解。

2. 指定房间录制

在未开启全局录制的情况下，您依然可以对个别重要的视频流开启录制，操作方法是在调用 Web 端 EXEStarter.js 中的 createExeAsRoom 接口时，将 custom 中的 record参数为true。

示例代码：

```
EXEStarter.createExeAsRoom({
userdata: {
userID: accountInfo.userID,
userName: accountInfo.userName,
```

```
sdkAppID: accountInfo.sdkAppID,
accType: accountInfo.accountType,
userSig: accountInfo.userSig,
},
roomdata: {
serverDomain: "https://room.qcloud.com/",
roomAction: object.data.roomAction,
roomID: object.data.roomID,
roomName: object.data.roomName,
roomTitle: object.data.roomTitle,
roomLogo: "http://liteav.myqcloud.com/windows/logo/liveroom_logo.png",
type: EXEStarter.StyleType.LiveRoom,
template: EXEStarter.Template.Template1V3,
},
custom: { //可选参数
record: true, // 后台录制当前视频流
},
success: function (ret) {
console.log('EXEStarter.openExeRoom 打开本地应用成功');
},
fail: function (ret) {
console.log('EXEStarter.openExeRoom 打开本地应用失败');
if (ret.errCode == -1) {
//本地应用未安装
}
},
})
```

这里有几点要特别说明：

- 目前录制功能默认使用的视频封装格式为 mp4（播放、下载及传输比较方便）。
- mp4 文件默认分片时长为2小时，只要中途不断流，2小时内的直播内容都将录制在同一个 mp4 文件中。

您可根据具体的业务需求进行选择，若您需要录制每一路视频，推荐选择全局录制。若您需要将混流画面录制在同一个文件中，推荐选择指定房间录制。

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，您可以结合自己的业务场景实现很多的扩展功能，比如：您可以将其追加到历史信息里，作为资料进行存档；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。

那么怎样才能拿到文件的地址呢？有如下三种解决方案：

1. 被动监听通知

您可以使用腾讯云的**事件通知服务**：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。

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.flv"
}
```

注意：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同一次直播，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stream_id、start_time 和 end_time 等信息）。

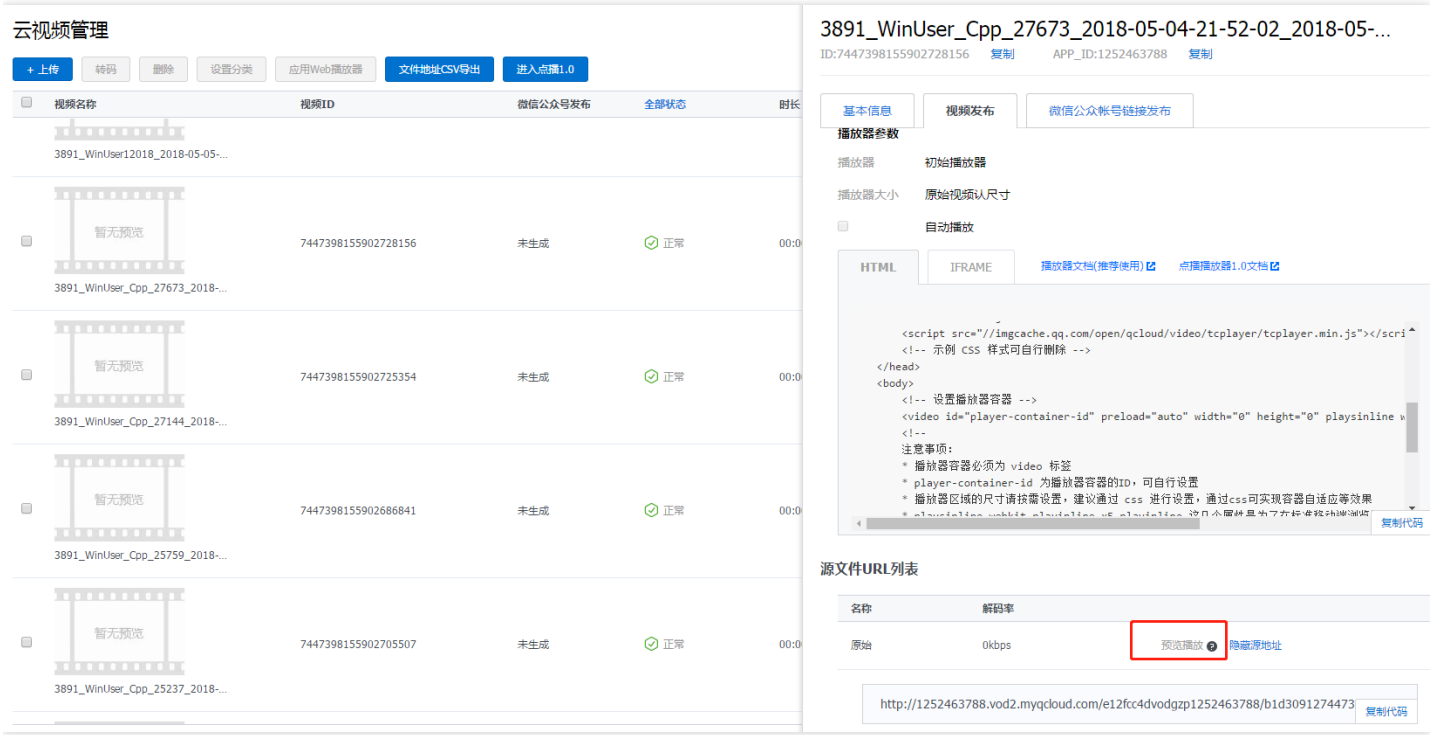
2. 主动查询文件

您可以通过腾讯云的文件查询接口（**Live_Tape_GetFilelist**）定时地查看是否有新的录制文件生成，不过这种方案在要查询的频道数特别多的时候，响应速度不理想，同时调用频率也不能太快（仅对刚结束的频道进行调用为宜），这种方案的实时性和可靠性不高，并不推荐频繁使用。

3. 查看视频管理中的文件

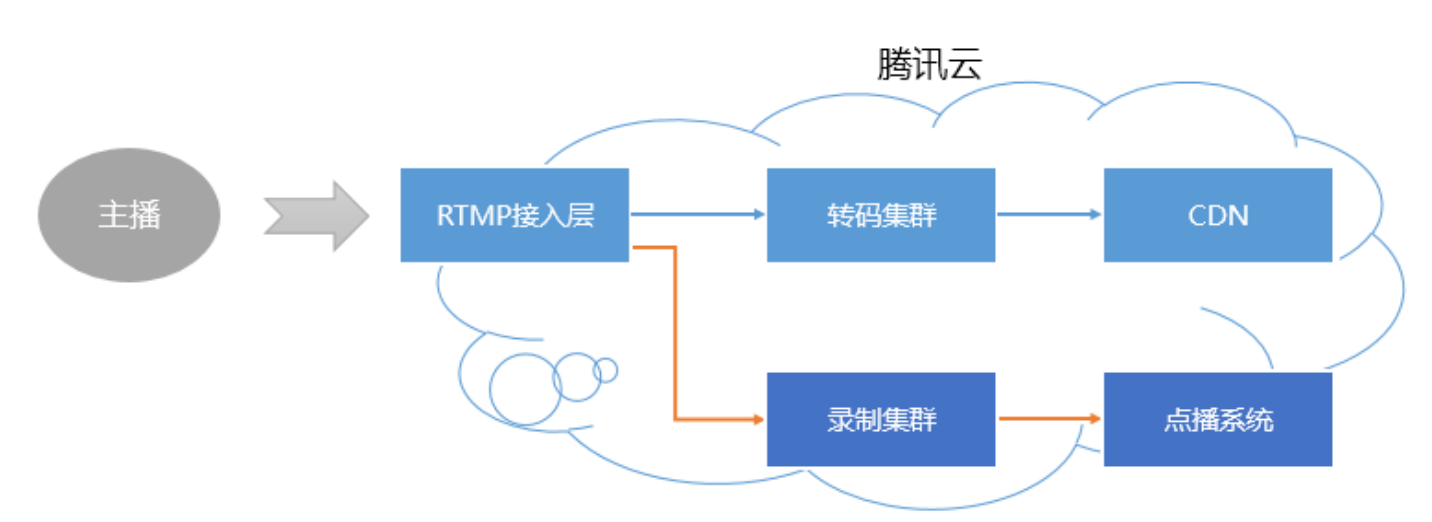
在点播控制台的 视频管理 里可以查询到所有的录制视频，适用于后续检索文件内容。可根据前缀搜索查询指定视频，如下图：

选择视频列表中对应文件后，可以在右侧看到具体视频内容，切换 tab 页至视频发布，点击选择显示源地址，即可得到文件地址。如下图：



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。

一旦直播流断中断，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。

2. 一次直播会有几个录制文件？

- 如果一次直播过程非常短暂，比如只有不到 1 秒钟时间，那么可能是没有录制文件的。
- 如果一次直播时间不算长——小于7天（hls 格式）或者小于 mp4、flv 格式的分片时长，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7天（hls 格式）或者超过 mp4、flv 格式的分片时长，那么会强制进行分片，分片的原因是避免过长的文件在分布式系统中流转的时间不确定性。
- 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。

3. 如何把碎片拼接起来？

目前腾讯云支持使用云端 API 接口拼接视频分片，API 详细用法可以参考 视频拼接 。

常见问题

常见问题

最近更新时间：2018-07-10 14:28:09

为何 <live-pusher> 和 <live-player> 不能工作？

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 个人账号和企业账号的小程序暂时只开放如下表格中的类目：
- 符合类目要求的小程序，需要在小程序管理后台的 “设置 - 接口设置” 中自助开通该组件权限。

主类目	子类目
【社交】	直播
【教育】	在线教育
【医疗】	互联网医院，公立医院
【政务民生】	所有二级类目
【金融】	基金、信托、保险、银行、证券/期货、非金融机构自营小额贷款、征信业务、消费金融

注意：如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

如何选用分辨率比例？

分辨率比例通过 aspect 设置，目前有 3：4 和 9：16 两种，根据当前推流与播放画面在手机上的显示区域比例来设置。手机竖屏状态一般都是 9：16 比例，如果一个人推流满屏，则设置 9：16 ；如果双人并排展示，一个推流一个播放，显示区域为 3：4 比例，则设置 3：4。

各种场景如何选用 SD、HD、FHD、RTC 模式，最小最大码率如何设置？

模式选择及码率设置请参考 <live-pusher> 标签参数设置部分。

推流及播放关键事件如何监听？

推流事件监听请参考 <live-pusher> 标签内部事件部分。
播放事件监听请参考 <live-player> 标签内部事件部分。

部分手机 <live-push> 标签会插入失败？

目前这是一个已知问题，下个版本会修复，暂时没有办法捕获这种异常提示用户开启麦克风与相机权限。
客户在开发过程中如果遇到 log 提示：“insertLivePusher:fail:system permission denied” 表明没有麦克风与相机权限，请在设置项里检查麦克风与相机权限，确保麦克风与相机权限已经打开。

小程序 <live-play> 标签偶现黑屏或者播放失败？

请先了解微信小程序 Page 生命周期，参考 小程序页面生命周期。

小程序 Page 生命周期，onLoad 只做数据加载还未做页面渲染，此时 <live-push>、<liveplay> 标签还未创建完成，获取或者调用 livepushercontext、liveplayercontext 的方法行为不确定。onReady 表示页面已经加载，完成初次渲染，跟 <live-pusher>、<live-player>标签相关的操作都需要放在 onReady 里面实现。
举例说明：

```
onReady: function () {
var self = this;
this.data.videoContext = wx.createLivePlayerContext("video-livePlayer");
this.setData({
playUrl: "rtmp://live.hkstv.hk.lxdns.com/live/hks",
}, function () {
self.data.videoContext.stop();
self.data.videoContext.play();
})
},
```

<live-player> 标签设置 style=”display:block” 出现 log 叠加如何解决？

此问题为已知的问题，如果想通过style的display属性来控制<live-player>标签的显示与隐藏，推荐隐藏设置为style=”display:none” ，显示的时候display采用默认值不做任何修改。

8. 使用同一个 <live-player> 标签播放不同的 URL 不生效如何解决？

当模式为 autoplay 时，playUrl 变更目前不会自动播放；非 autoplay 模式时改变 playUrl 之后需要调用 play 函数。因此不论哪种模式当改变 playUrl 播放另外一个流地址时，推荐的做法为：

```
onPlay: function () {
var self = this;
this.data.videoContext = wx.createLivePlayerContext("video-livePlayer");
this.setData({
playUrl: "rtmp://live.hkstv.hk.lxdns.com/live/hks",
}, function () {
self.data.videoContext.stop();
self.data.videoContext.play();
})
},
```

请注意这里一定要在 setData 的回调里面调用 videoContext 的方法，同时要先调用 stop 再调用 start。

<live-player>标签动态设置 mode 属性不生效如何解决？

目前动态设置 mode 属性没有做到动态生效，需要重新 stop 再 start，具体请参考如下使用方式：

```
onChangeMode: function () {
var self = this;
this.data.videoContext = wx.createLivePlayerContext("video-livePlayer");
if (this.data.mode == "live") {
this.setData({ mode: "RTC"}, function () {
self.data.videoContext.stop();
self.data.videoContext.play();
});
}
else {
this.setData({ mode: "live"}, function () {
self.data.videoContext.stop();
self.data.videoContext.play();
});
}
},
```

Android 小程序 <live-player> 标签在 RTC 模式下部分手机设置 muted 属性为何不能静音？

这是一个已知的 BUG，我们会尽快在 SDK 中修复，并赶上微信下一个版本的发布。

<live-pusher> 标签动态修改 enable-camera 属性不生效如何解决？

目前 enable-camera 属性不支持动态生效，需要提前设置，即不支持在推流的过程中动态开启/关闭摄像头，必须在推流之前设置好这个属性。如果需要动态生效推荐的使用方式如下：

```
onEnableCamera: function () {
var self = this;
this.data.videoContext = wx.createLivePusherContext("video-livePusher");
this.setData({
enable-camera: "true"
},function () {
self.data.videoContext.stop();
self.data.videoContext.start();
});
},
```

建议在页面 onUnload 里面对 <live-pusher> 与<live-player> 标签做清理

具体请参考如下：

```
onUnload: function () {
self.data.pusherContext && self.data.pusherContext.stop();
self.data.playerContext && self.data.playerContext.stop();
},
```

<live-pusher> 或 <live-player> 标签上面叠加 cover-view 时不推荐修改标签的大小

当前<live-pusher>或<live-player>大小变化时， iOS版本小程序对叠加在上面的cover-view不能追随变化，界面显示不可预期, 因此当在<live-pusher>或<live-player>标签上面叠加cover-view时，不要动态修改<live-pusher>或<live-player>标签的大小。

Demo 源码里面的 <cameraview> 有什么作用？

由于仅仅依靠 <live-pusher> 和 <live-player> 两个标签实现实时音视频通话（尤其是多人视频通话），需要大量的开发工作。所以我们基于微信小程序原生的 **自定义组件** 能力，将复杂的状态管理和业务逻辑封装在了这个叫做 <cameraview> 的自定义标签中，您可以通过源码包中 `doubleroom\room\room.js` 里的代码看到如何使用 <cameraview> 和 <live-player> 快速构建一个双人音视频通话功能。