实时音视频

旧版文档

腾讯云

文档目录

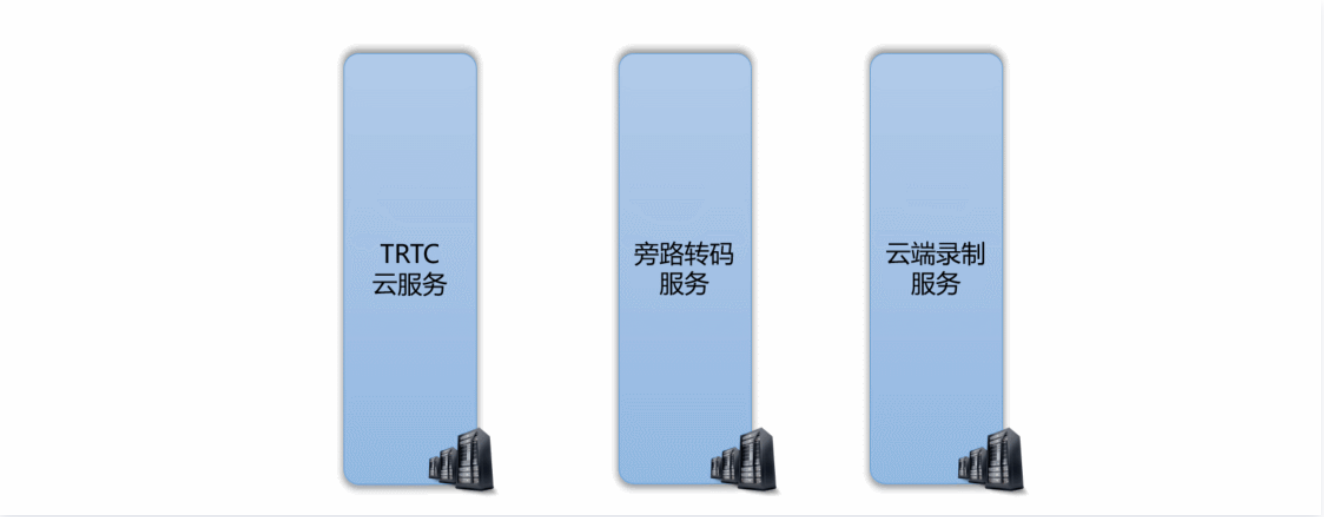
旧版文档
云端录制与回放(旧)

最近更新时间：2024-12-10 11:16:52

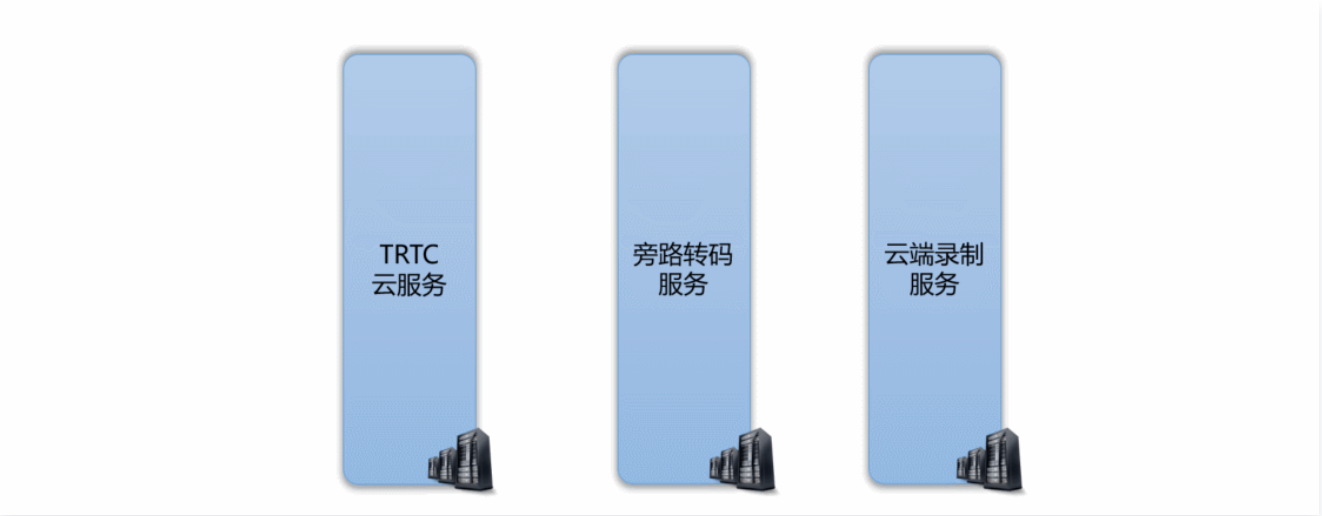
适用场景

在远程教育、秀场直播、视频会议、远程定损、金融双录、在线医疗等应用场景中，考虑取证、质检、审核、存档和回放等需求，常需要将整个视频通话或互动直播过程录制和存储下来。
TRTC 的云端录制，可以将房间中的每一个用户的音视频流都录制成一个独立的文件：

> **注意：**
> 本篇云端录制功能说明仅适用于当前应用（sdkappid）的云端录制功能为**旧版云端录制**功能时（如何区分新、旧版云端录制功能），若您应用的云端录制功能为新版时，请参见 新版云端录制 发起云端录制功能。



也可以将房间中的多路音视频先进行 云端混流，再将混合后的音视频流录制成一个文件：



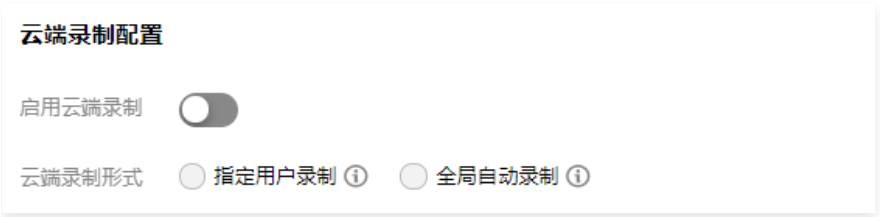
控制台指引

开通录制服务

1. 登录实时音视频控制台，在左侧导航栏选择 应用管理 。
2. 单击目标应用所在行的**功能配置**，进入功能配置页卡。如果您还没有创建过应用，可以单击**创建应用**，填写应用名称，单击**确定**创建一个新的应用。
3. 单击**启用云端录制**右侧的，会弹出云端录制的设置页面。

选择录制形式

TRTC 的云端录制服务提供了两种不同的录制形式：“全局自动录制”和“指定用户录制”：



- **全局自动录制**
  每一个 TRTC 房间中的每个用户的音视频上行流都会被自动录制下来，录制任务的启动和停止都是自动的，不需要您额外操心，比较简单和易用。具体的使用方法请阅读 方案一：全局自动录制 。
- **指定用户录制**
  您可以指定只录制一部分用户的音视频流，这需要您通过客户端的 SDK API 或者服务端的 REST API 进行控制，需要额外的开发工作量。具体的使用方法请阅读 方案二：指定用户录制（SDK API） 和 方案三：指定用户录制（REST API） 。

选择文件格式

云端录制支持 HLS、MP4、FLV 和 AAC 四种不同的文件格式，我们以表格的形式列出四种不同格式的差异和适用场景，您可以结合自身业务的需要进行选择：

参数	参数说明
文件类型	支持以下文件类型： • HLS：该文件类型支持绝大多数浏览器在线播放，适合视频回放场景。选择该文件类型时，支持断点续录且不限制单个文件最大时长。 • FLV：该文件类型不支持在浏览器在线播放，但该格式简单容错性好。如果无需将录制文件存储在云点播平台，可以选择该文件类型，录制完成后立刻下载录制文件并删除源文件。 • MP4：该文件类型支持在 Web 浏览器在线播放，但此格式容错率差，视频通话过程中的任何丢包都会影响最终文件的播放质量。 • AAC：如果只需录制音频，可以选择该文件类型。
单个文件的最大时长（分钟）	• 根据实际业务需求设置单个视频文件的最大时长限制，超过长度限制后系统将会自动拆分视频文件。单位为分钟，取值范围1 – 120。 • 当文件类型设置为HLS时，不限制单个文件的最大时长，即该参数无效。
文件保存时长（天）	根据实际业务需求设置视频文件存储在云点播平台的天数。单位为天，取值范围0 – 1500，到期后文件将被点播平台自动删除且无法找回， 0表示永久存储。
续录超时时长（秒）	• 默认情况下，若通话（或直播）过程因网络波动或其他原因被打断，录制文件会被切断成多个文件。 • 如果需要实现“一次通话（或直播）只产生一个回放链接”，可以根据实际情况设置续录超时时长，当打断间隔不超过设定的续录超时时长时，一次通话（或直播）只会生成一个文件，但需要等待续录时间超时后才能收到录制文件。 • 单位为秒，取值范围1 – 1800，0表示断点后不续录。

说明：
HLS 支持最长三十分钟的续录，可以做到“一堂课只产生一个回放链接”，且支持绝大多数浏览器的在线观看，非常适合在线教育场景中的视频回放场景。

选择存储位置

TRTC 云端录制文件会默认存储于腾讯云点播服务上，如果您的项目中多个业务公用一个腾讯云点播账号，可能会有录制文件隔离的需求。您可以通过腾讯云点播的“子应用”能力，将 TRTC 的录制文件与其他业务区分开。

- **什么是点播主应用和子应用?**
  主应用和子应用是云点播上的一种资源划分的方式。主应用相当于云点播的主账号，子应用可以创建多个，每一个子应用相当于主账号下面的一个子账号，拥有独立的资源管理，存储区域可以跟其他的子应用相互隔离。
- **如何开启点播服务的子应用?**
  您可以根据文档 “如何开启点播子应用” 添加新的子应用，这一步需要您跳转到 点播控制台 中进行操作。

设置录制回调

- **录制回调地址：**
  如果您需要实时接收到新文件的 落地通知，可在此处填写您的服务器上用于接收录制文件的回调地址，该地址需符合 HTTP（或 HTTPS）协议。当新的录制文件生成后，腾讯云会通过该地址向您的服务器发送通知。

- **录制回调密钥：**
  回调密钥用于在接收回调事件时生成签名鉴权，该密钥需由大小写字母及数字组成，且不得超过32个字符。相关使用请参见 录制事件参数说明 。

> **说明**
> 详细的录制回调接收和解读方案请参考文档后半部分的： 接收录制文件 。

录制控制方案

TRTC 提供了三种云端录制的控制方案，分别是 全局自动录制 、指定用户录制（由 SDK API 控制） 和 指定用户录制（由 REST API 控制） 。对于其中的每一种方案，我们都会详细介绍：

- 如何在控制台中设定使用该方案?

- 如何开始录制任务?
- 如何结束录制任务?
- 如何将房间中的多路画面混合成一路?
- 录制下来的文件会以什么格式命名?
- 支持该种方案的平台有哪些?

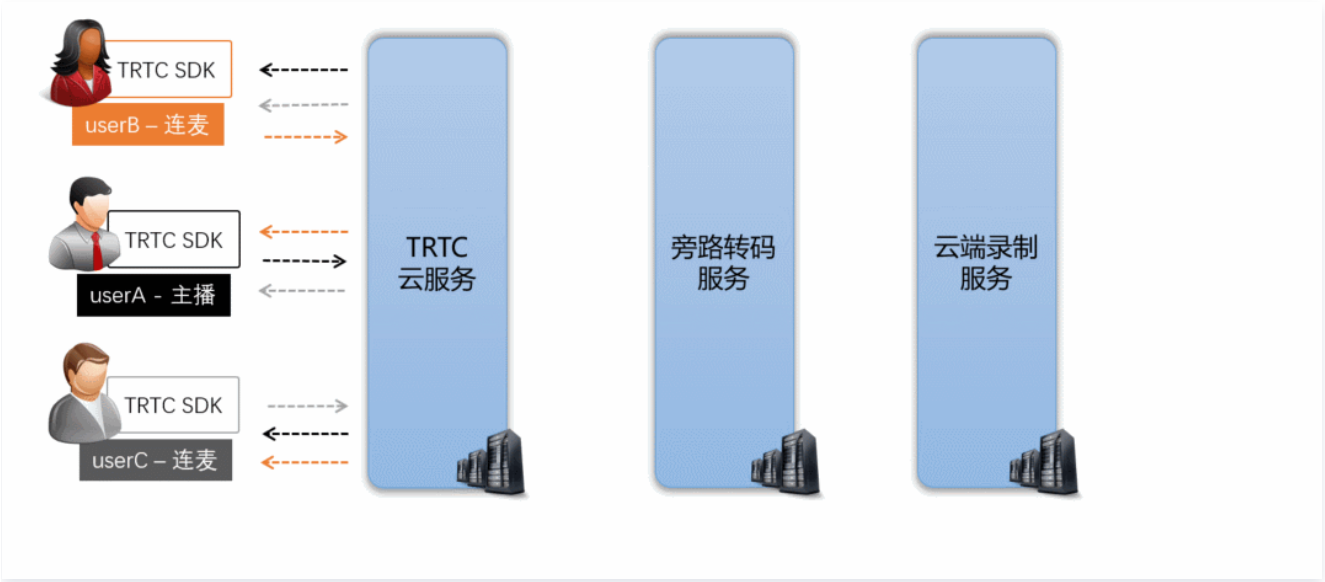
方案一：全局自动录制

- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“全局自动录制”。
- **录制任务的开始**
  TRTC 房间中的每一个用户的音视频流都会被自动录制成文件，无需您的额外操作。
- **录制任务的结束**
  自动停止。即每个主播在停止音视频上行后，该主播的云端录制即会自行停止。如果您在 选择文件格式 时设置了“续录时间”，则需要等待续录时间超时后才能收到录制文件。
- **多路画面的混合**
  全局自动录制模式下的云端混流有两种方案，即“服务端 REST API 方案” 和 “客户端 SDK API 方案”，两套方案请勿混合使用：
  - 服务端 REST API 混流方案：需要由您的服务器发起 API 调用，不受客户端平台版本的限制。
  - 客户端 SDK API 混流方案：可以直接在客户端发起混流，目前支持 iOS、Android、Windows、Mac 、Web 等平台，暂不支持微信小程序。
- **录制文件的命名**
  - 如果主播在进房时指定了 userDefineRecordId 参数，则录制文件会以 `userDefineRecordId_streamType_开始时间_结束时间` 来命名（streamType 有 main 和 aux 两个取值，main 代表主路，aux 代表辅路，辅路通常被用作屏幕分享）；
  - 如果主播在进房时没有指定 userDefineRecordId 参数，但指定了 streamId 参数，则录制文件会以 `streamId_开始时间_结束时间` 来命名；
  - 如果主播在进房时既没有指定 userDefineRecordId 参数，也没有指定 streamId 参数，则录制文件会以 `sdkappid_roomid_userid_streamType_开始时间_结束时间` 来命名（streamType 有 main 和 aux 两个取值，main 代表主路，aux 代表辅路，辅路通常被用作屏幕分享）。
- **已经支持的平台**
  由您的服务端控制，不受客户端平台限制。

方案二：指定用户录制（SDK API）

通过调用 TRTC SDK 提供的一些 API 接口和参数，即可实现云端混流、云端录制和旁路直播三个功能：

云端能力	如何开始?	如何停止?
云端录制	进房时指定参数 `TRTCParams` 中的 `userDefineRecordId` 字段	主播退房时自动停止
云端混流	调用 SDK API setMixTranscodingConfig() 启动云端混流	发起混流的主播退房后，混流会自动停止，或中途调用 setMixTranscodingConfig() 并将参数设置为 `null/nil` 手动停止
旁路直播	进房时指定参数 `TRTCParams` 中的 `streamId` 字段	主播退房时自动停止



- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“指定用户录制”。
- **录制任务的开始**
  主播在进房时指定进房参数 TRTCParams 中的 userDefineRecordId 字段，之后该主播的上行音视频数据即会被云端录制下来，不指定该参数的主播不会触发录制任务。

```
// 示例代码：指定录制用户 rexchang 的音视频流，文件 id 为 1001_rexchang
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
TRTCParams *param = [[TRTCParams alloc] init];
param.sdkAppId = 1400000123;      // TRTC 的 SDKAppID，创建应用后可获得
param.roomId    = 1001;           // 房间号
param.userId    = @"rexchang";    // 用户名
param.userSig   = @"xxxxxxxx";    // 登录签名
param.role      = TRTCRoleAnchor; // 角色：主播
param.userDefineRecordId = @"1001_rexchang";  // 录制 ID，即指定开启该用户的录制。
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE]; // 请使用 LIVE 模式
```

- **录制任务的结束**
  自动停止，当进房时指定 userDefineRecordId 参数的主播在停止音视频上行后，云端录制会自行停止。如果您在 选择文件格式 时设置了“续录时间”，则需要等待续录时间超时后才能收到录制文件。
- **多路画面的混合**
  您可以通过调用 SDK API setMixTranscodingConfig() 将房间中其他用户的画面和声音混合到当前用户的这一路音视频流上。关于这一部分详细介绍，可以阅读文档：云端混流转码。

> ⚠ **注意：**
> 在一个 TRTC 房间中，只由一个主播（推荐是开播的主播）来调用 `setMixTranscodingConfig` 即可，多个主播调用可能会出现状态混乱的错误。

- **录制文件的命名**
  录制文件会以 `userDefineRecordId_开始时间_结束时间` 的格式来命名。
- **已经支持的平台**
- 支持 iOS、Android、Windows、Mac、Electron、Web 等终端发起录制控制，暂不支持微信小程序端发起控制。

方案三：指定用户录制（REST API）

TRTC 的服务端提供了一对 REST API（ StartMCUMixTranscode 和 StopMCUMixTranscode ）用于实现云端混流、云端录制和旁路直播三个功能：

云端能力	如何开始?	如何停止?
云端录制	调用 StartMCUMixTranscode 时指定 `OutputParams.RecordId` 参数即可开始录制	自动停止，或中途调用 StopMCUMixTranscode 停止
云端混流	调用 StartMCUMixTranscode 时指定 `LayoutParams` 参数可设置布局模板和布局参数	所有用户退房后自动停止，或中途调用 StopMCUMixTranscode 手动停止
旁路直播	调用 StartMCUMixTranscode 时指定 `OutputParams.StreamId` 参数可启动到 CDN 的旁路直播	自动停止，或中途调用 StopMCUMixTranscode 停止

说明：
由于这对 REST API 控制的是 TRTC 云服务中的核心混流模块 MCU，并将 MCU 混流后的结果输送给录制系统和直播 CDN，因此 API 的名字被称为 `Start/StopMCUMixTranscode` 。因此，从功能角度上来说， `Start/StopMCUMixTranscode` 不仅仅可以实现混流的功能，也可以实现云端录制和旁路直播 CDN 的功能。

- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“指定用户录制”。
- **录制任务的开始**
  由您的服务器调用 StartMCUMixTranscode ，并指定 `OutputParams.RecordId` 参数即可启动混流和录制。

```
// 代码示例：通过 REST API 启动云端混流和云端录制任务
https://trtc.tencentcloudapi.com/?Action=StartMCUMixTranscode
&SdkAppId=1400000123
&RoomId=1001
&OutputParams.RecordId=1400000123_room1001
&OutputParams.RecordAudioOnly=0
&EncodeParams.VideoWidth=1280
&EncodeParams.VideoHeight=720
&EncodeParams.VideoBitrate=1560
&EncodeParams.VideoFramerate=15
```

```
&EncodeParams.VideoGop=3
&EncodeParams.BackgroundColor=0
&EncodeParams.AudioSampleRate=48000
&EncodeParams.AudioBitrate=64
&EncodeParams.AudioChannels=2
&LayoutParams.Template=1
&<公共请求参数>
```

> **注意**
> - 该 REST API 需要在房间中至少有一个用户进房（enterRoom）成功后才有效。
> - 使用 REST API 不支持单流录制，如果您需要单流录制，请选择 方案一 或者 方案二 。

- **录制任务的结束**
  自动停止，您也可以中途调用 StopMCUMixTranscode 停止混流和录制任务。
- **多路画面的混合**
  在调用 StartMCUMixTranscode 时同时指定 `LayoutParams` 参数即可实现云端混流。该 API 支持在整个直播期间多次调用，即您可以根据需要修改 `LayoutParams` 参数并再次调用该 API 来调整混合画面的布局。但需要注意的是，您需要保持参数 `OutputParams.RecordId` 和 `OutputParams.StreamId` 在多次调用中的一致性，否则会导致断流并产生多个录制文件。

> **说明:**
> 关于云端混流的详细介绍，具体请参见 云端混流转码 。

- **录制文件的命名**
  录制文件会以调用 StartMCUMixTranscode 时指定的 `OutputParams.RecordId` 参数来命名，命名格式为 `OutputParams.RecordId_开始时间_结束时间` 。
- **已经支持的平台**
  由您的服务端控制，不受客户端平台的限制。

查找录制文件

在开启录制功能以后，TRTC 系统中录制下来的文件就能在腾讯云点播服务中找到。您可以直接在云点播控制台手动查找，也可以由您的后台服务器使用 REST API 进行定时筛选：

方式一：在点播控制台手动查找

1. 登录 云点播控制台 ，在左侧导航栏选择**媒资管理**。
2. 单击列表上方的**前缀搜索**，选择**前缀搜索**，在搜索框输入关键词，例如 `1400000123_1001_rexchang_main` ，单击搜索按钮，将展示视频名称前缀相匹配的视频文件。
3. 您可以根据创建时间筛选所需的目标文件。

方式二：通过点播 REST API 查找

腾讯云点播系统提供了一系列 REST API 来管理其上的音视频文件，您可以通过 搜索媒体信息 这个 REST API 来查询您在点播系统上的文件。您可以通过请求参数表中的 `Text` 参数进行模糊匹配，也可以根据 `StreamId` 参数进行精准查找。
REST 请求示例：

```
https://vod.tencentcloudapi.com/?Action=SearchMedia
&StreamId=stream1001
&Sort.Field=CreateTime
&Sort.Order=Desc
&<公共请求参数>
```

接收录制文件

除了 查找录制文件，您还可以通过配置回调地址，让腾讯云主动把新录制文件的消息推送给您的服务器。
房间里的最后一路音视频流退出后，腾讯云会结束录制并将文件转存到云点播平台，该过程大约默认需要30秒至2分钟（若您设置了续录时间为300秒，则等待时间将在默认基础上叠加300秒）。转存完成后，腾讯云会通过您在 设置录制回调 中设置的回调地址（HTTP/HTTPS）向您的服务器发送通知。
腾讯云会将录制和录制相关的事件都通过您设置的回调地址推送给您的服务器，回调消息示例如下图所示：


```
{
    "app": "8888.livepush.myqcloud.com",
    "appid": 1234567890,
    "appname": "trtc_1400000123",
    "channel_id": "1400000123_1001_rexchang_main",
    "duration": 39,
    "end_time": 1581926501,
    "end_time_usec": 817545,
    "event_type": 100,
    "file_format": "mp4",
    "file_id": "5285890798833426017",
    "file_size": 3337482,
    "media_start_time": 0,
    "record_bps": 0,
    "record_file_id": "5285890798833426017",
    "start_time": 1581926464,
    "start_time_usec": 588890,
    "stream_id": "1400000123_1001_rexchang_main",
    "stream_param": "txSecret=ecd19683f6cf1a7615f13670e0834de1&txTime=70d4653d&from=interactive&client_business_type=0&
                     sdkappid=1400000123&sdkapptype=1&groupid=1001&userid=cmV4Y2hhbmc=&ts=5e4a483c&userdefinerecordid=my
                     testfile&tinyid=1441152145405491 89&roomid=2294546&cliRecoId=0&trtcclientip=222.248.237.246&useMixPl
                     ayer=1",
    "task_id": "1802569503626226707",
    "video_id": "1234567890_46ac1271218249118752d57423120300",
    "video_url": "http://1234567890.vod2.myqcloud.com/5022b150vodcq1234567890/39e578f12280795898833426017/f0.mp4"
}
```


您可以通过下表中的字段来确定当前回调是对应的哪一次通话（或直播）：

序号	字段名	说明
1	event_type	消息类型，当 event_type 为100时，表示该回调消息为录制文件生成的消息。
2	stream_id	即直播 CDN 的 streamId，您可以在进房时通过设置 TRTCParams 中的 streamId 字段指定（推荐），也可以在调用 TRTCCloud 的 startPublishing 接口时通过参数 streamId 来指定。
3	stream_param.userid	用户名的 Base64 编码。
4	stream_param.userdefinerecordid	自定义字段，您可以通过设置 TRTCParams 中的 userDefineRecordId 字段指定。
5	video_url	录制文件的观看地址，可以用于 点播回放 。

> **说明：**
> 更多回调字段说明，请参见 云直播-录制事件通知 。

删除录制文件

腾讯云点播系统提供了一系列 REST API 来管理其上的音视频文件，您可以通过 删除媒体 API 删除某个指定的文件。
REST 请求示例：

```
https://vod.tencentcloudapi.com/?Action=DeleteMedia
&FileId=5285890798866415058 7
&<公共请求参数>
```

回放录制文件

在线教育等场景中，通常需要在直播结束后多次回放录制文件，以便充分利用教学资源。

选择文件格式（HLS）

在 设置录制格式 中选择文件格式为 HLS。
HLS 支持最长三十分钟的断点续录，可以做到“一场直播（或一堂课）只产生一个回放链接”，且 HLS 文件支持绝大多数浏览器在线播放，非常适合视频回放场景。

获取点播地址（video_url）

在 接收录制文件 时，可以获取回调消息中 video_url 字段，该字段为当前录制文件在腾讯云的点播地址。

对接点播播放器

根据使用平台对接点播播放器，具体操作参考如下：

- iOS 平台
- Android 平台
- Web 浏览器

> ⚠ **注意：**
> 建议使用 全功能版 TRTC SDK，全功能版集合了 超级播放器（Player+）、直播 SDK 等功能，由于底层模块的高度复用，集成全功能版的体积增量要小于同时集成两个独立的 SDK，并且可以避免符号冲突（symbol duplicate）的困扰。

相关费用

云端录制与回放功能使用到的功能包括：云服务资源（包括云端录制服务、云点播的回放文件存储与处理、云点播的播放服务等），和终端 SDK 播放点播视频的能力。可能会根据实际需求产生以下费用。

云服务消耗

云服务消耗包括了云端录制产生的费用和回放文件产生的消耗费用。您可根据实际需求进行使用。

> ⓘ **说明：**
> 本文中的价格为示例，仅供参考。若价格与实际不符，请以 云端录制计费说明、云直播 和 云点播 的定价为准。

- **录制费用：转码或转封装产生的计算费用**
  由于录制需要进行音视频流的转码或转封装，会产生服务器计算资源的消耗，因此需要按照录制业务对计算资源的占用成本进行收费。

  > ⚠ **注意：**
  > - 自2020年7月1日起首次在 TRTC 控制台创建应用的腾讯云账号，使用云端录制功能后产生的录制费用以 云端录制计费说明 为准。
  > - 在2020年7月1日之前已经在 TRTC 控制台创建过应用的腾讯云账号，无论是在2020年7月1日之前还是之后创建的应用，使用云端录制功能产生的录制费用均默认继续延用**直播录制**的计费规则。

  直播录制计费的计算方法是按照并发录制的路数进行收费，并发数越高录制费用越高，具体计费说明请参见 云直播 > 直播录制 。

  计算公式：
  录制有效天数占比 = 一个自然月使用录制功能的天数 / 当月总天数。
  录制费用 = 一个自然月录制并发路数峰值 × 录制有效天数占比 × 录制路数单价。

  例如，您4月份有1000个主播，如果在晚高峰时，最多同时有500路主播的音视频流需要录制，同时在4月份共有6天使用了录制功能（有效天数占比为6/30）。假设录制单价为30元/路/月，那么4月份总录制费用为 `500路 × 0.2 / × 30元/路/月 = 3000元/月` 。
  如果您在 设置录制格式 时同时选择了两种录制文件，录制费用和存储费用都会 × 2，同理，选择三种文件时录制费用和存储费用会 × 3。
  如非必要，建议只选择需要的一种文件格式，可以大幅节约成本。

- **转码费用：如开启混流录制则会产生该费用**
  如果您启用了混流录制，由于混流本身需要进行解码和编码，所以还会产生额外的混流转码费用。混流转码根据分辨率大小和转码时长进行计费，主播用的分辨率越高，连麦时间（通常在连麦场景才需要混流转码）越长，费用越高，具体费用计算请参见 直播转码 。

> 例如，您通过 setVideoEncoderParam() 设置主播的码率（videoBitrate）为1500kbps，分辨率为720P。如果有一位主播跟观众连麦了一个小时，连麦期间开启了 云端混流，那么产生的转码费用为 `0.0325元/分钟 × 60分钟 = 1.95元` 。

- **回放文件存储费用：云端视频存储服务产生的存储费用**
- 录制出的视频文件存放于云点播服务，会使用云点播的云端存储服务。由于存储本身会产生磁盘资源的消耗，因此需要按照存储的资源占用进行收费。存储的时间越久费用也就越高，因此如无特殊需要，您可以将文件的存储时间设置的短一些来节省费用，或者将文件存放在自己的服务器上。存储费用可以选择 视频存储（日结）价格 进行日结计算，也可以购买 存储资源包 。

> 例如，您通过 setVideoEncoderParam() 设置主播的码率（videoBitrate）为1000kbps，录制该主播的直播视频（选择一种文件格式），录制一小时大约会产生一个 `(1000 / 8)KBps × 3600秒 = 450000KB = 0.45GB` 大小的视频文件，该文件每天产生的存储费用约为 `0.45GB × 0.0048 元/GB/日 = 0.00216元` 。

- **观看回放费用：云点播视频进行分发播放产生的播放费用**
- 如果您录制的视频文件要被用于回看播放，会使用云点播的 CDN 播放功能。由于观看本身会产生 CDN 流量消耗，因此需要按照点播的价格进行计费，默认按流量收费。观看的人数越多费用越高，观看费用可以选择 视频加速（日结）价格 进行日结计算，也可以购买 流量套餐包 。

> 例如，您通过云端录制产生了一个1GB大小的文件，且有1000位观众从头到尾完整地观看了视频，大约会产生1TB的点播观看流量，那么按照阶梯价格表，1000位观众就会产生 `1000 × 1GB × 0.23元/GB = 230元` 的费用，按照流量套餐包则是175元。
> 如果您选择从腾讯云下载文件到您的服务器上，也会产生一次很小的点播流量消耗，并且会在您的月度账单中有所体现。

SDK 播放授权

实时音视频（TRTC）全功能版本 SDK 提供了功能全面性能强大的视频播放能力，可轻松配合云点播实现视频播放功能。移动端 SDK 在10.1及以上的版本可通过获取指定 License 以解锁视频播放能力。

> ⚠ **注意：**
> TRTC 的播放能力无需 License 授权。

您可直接 购买视频播放 License，或通过 购买的云点播流量包 免费获赠视频播放 License 或 短视频 License，两种 License 均可用于解锁 SDK 的视频播放功能。并且点播资源包可以抵扣云点播的播放产生的日结流量，详细说明请参见 云点播预付费资源包 。
License 计费说明参见 腾讯云视立方 License，License 购买完成后可参考 License 操作指引 进行新增和续期等操作。

云端混流转码(旧)

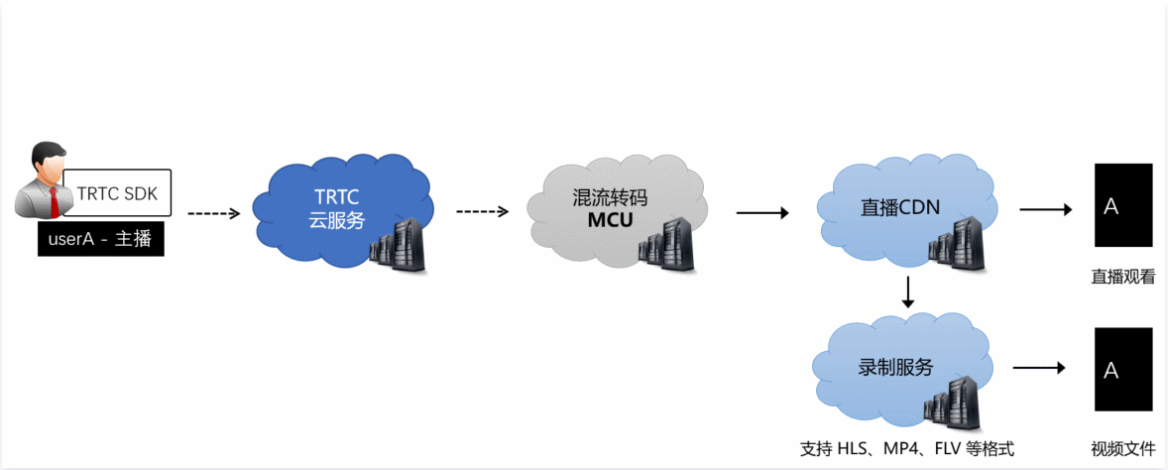
最近更新时间：2024-12-03 16:33:13

适用场景

在 CDN 直播观看 和 云端录制回放 等应用场景中，常需要将 TRTC 房间里的多路音视频流混合成一路，您可以使用腾讯云服务端的 MCU 的混流转码集群完成该项工作。MCU 集群能将多路音视频流进行按需混合，并将最终生成的视频流分发给直播 CDN 和云端录制系统。

云端混流有两种控制方式：

- **方案一**：使用服务端 REST 接口 StartMCUMixTranscode（数字房间号版本 / 字符串房间号版本）和 StopMCUMixTranscode（数字房间号版本 / 字符串房间号版本）进行控制，该接口还可以同时支持启动 CDN 观看和发起 旧版云端录制功能 。
- **方案二**：使用客户端 TRTC SDK 的 setMixTranscodingConfig 接口进行控制，其原理如下图：

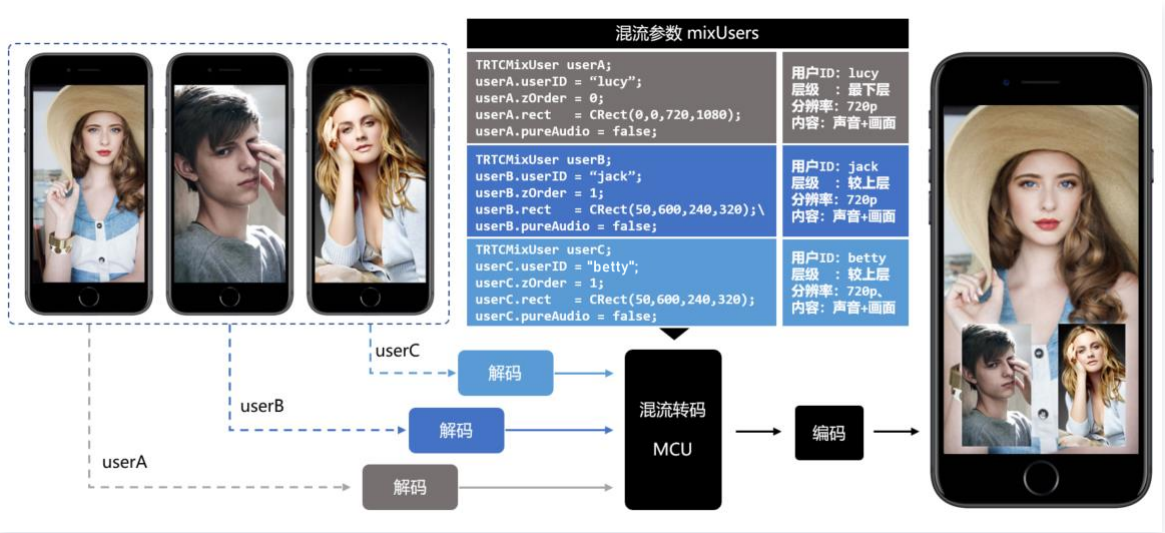


> **注意**
> 方案二支持 iOS、Android、Windows、Mac、Electron、Flutter 和 Web 平台的 SDK，如果您希望在微信小程序上也实现混流功能，请使用方案一。

原理解析

云端混流包含解码、混合和再编码三个过程：

- **解码**：MCU 需要将多路音视频流进行解码，包括视频解码和音频解码。
- **混合**：MCU 需要将多路画面混合在一起，并根据来自 SDK 的混流指令实现具体 的排版方案。同时，MCU 也需要将解码后的多路音频信号进行混音处理。
- **编码**：MCU 需要将混合后的画面和声音进行二次编码，并封装成一路音视频流，交给下游系统（例如直播和录制）。



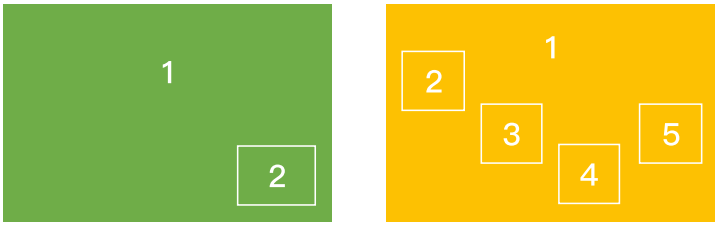
方案一：服务端 REST API 混流方案

启动混流

由您的服务器调用 REST API StartMCUMixTranscode 可以启动云端混流，对于此 API 您需关注如下细节：

步骤1：设置画面排版模式（必需）

通过 `StartMCUMixTranscode` 中的 LayoutParams 参数，可以设置如下几种排版模式：



- **悬浮模板（LayoutParams.Template = 0）**
  - 第一个进入房间的用户的视频画面会铺满整个屏幕，其他用户的视频画面从左下角依次水平排列，显示为小画面。
  - 最多4行，每行最多4个，小画面悬浮于大画面之上。
  - 最多支持1个大画面和15个小画面。
  - 如果用户只发送音频，仍然会占用画面位置。
- **九宫格模板（LayoutParams.Template = 1）**
  - 所有用户的视频画面大小一致，平分整个屏幕，人数越多，每个画面的尺寸越小。
  - 最多支持16个画面，如果用户只发送音频，仍然会占用画面位置。

- **屏幕分享模板（LayoutParams.Template = 2）**
  - 适合视频会议和在线教育场景的布局。
  - 屏幕分享（或者主讲的摄像头）始终占据屏幕左侧的大画面位置，其他用户依次垂直排列于右侧。
  - 需要通过 `LayoutParams.MainVideoUserId` 和 `LayoutParams.MainVideoStreamType` 这两个参数来指定左侧主画面的内容。
  - 最多两列，每列最多8个小画面。最多支持1个大画面和15个小画面。
  - 如果用户只发送音频，仍然会占用画面位置。
- **画中画模板（LayoutParams.Template = 3）**
  - 该模板以“一大一小，一前一后”的模式混合房间中的两路画面，即将房间中一路画面铺满整个屏幕，再将一路画面作为小画面悬浮在大画面之上，小画面的位置可以通过参数指定。
  - 您可以通过 `LayoutParams` 中的 `MainVideoUserId` 和 `MainVideoStreamType` 参数指定大画面这一路的用户 ID 和流类型。
  - 您可以通过 `LayoutParams` 中的 `SmallVideoLayoutParams` 参数指定小画面这一路的用户 ID、流类型和布局位置等信息。
  - 场景示例1：在线教育场景中，混合老师端摄像头（通常作为小画面）和老师端屏幕（通常作为大画面）两路画面，并混合课堂中学生的声音。
  - 场景示例2：1v1 视频通话场景中，混合远端用户画面（通常作为大画面）和本地用户画面（通常作为小画面）。
- **自定义模板（LayoutParams.Template = 4）**
  - 适用于需要自定义排布各路画面位置的场景，您可以通过 `LayoutParams` 中的 `PresetLayoutConfig` 参数（这是一个数组），预先设置各路画面的位置。
  - 您可以不指定 `PresetLayoutConfig` 参数中的 `UserId` 参数，排版引擎会根据进房的先后顺序，将进房的用户依次分配到 `PresetLayoutConfig` 数组中指定的各个位置上。
  - 如果 `PresetLayoutConfig` 数组中的某一个被指定了 `UserId` 参数，则排版引擎会预先给指定的用户预留好他/她在画面中的位置。
  - 如果用户只上行音频，不上行视频，该用户依然会占用画面位置。
  - 当 `PresetLayoutConfig` 数组中预设的位置被用完后，排版引擎将不再混合其他用户的画面和声音。

> ⚠ **注意：**
> 云端混流服务最多支持同时混合16路音视频流，如果用户只有音频也会被算作一路。

步骤2：设置混流编码参数（必需）

通过 `StartMCUMixTranscode` 中的 EncodeParams 参数，可以设置混流编码参数：

名称	描述	推荐值
AudioSampleRate	混流-输出流音频采样率	48000
AudioBitrate	混流-输出流音频码率，单位 kbps	64
AudioChannels	混流-输出流音频声道数	2
VideoWidth	混流-输出流宽，音视频输出时必填	自定义
VideoHeight	混流-输出流高，音视频输出时必填	自定义
VideoBitrate	混流-输出流码率，单位 kbps，音视频输出时必填	自定义
VideoFramerate	混流-输出流帧率，音视频输出时必填	15
VideoGop	混流-输出流 GOP，音视频输出时必填	3
BackgroundColor	混流-输出流背景色	自定义

步骤3：指定混合后的 streamID（必需）

- **OutputParams.StreamId**
  通过该参数您可以指定混合后的音视频流在直播 CDN 上的 streamID。不过只有在您已经开通了直播服务并配置了播放域名的情况下，才能通过 CDN 正常观看这条直播流。
- **OutputParams.PureAudioStream**
  如果您只希望做纯音频直播，可以设置 `OutputParams.PureAudioStream` 参数为 1，代表仅把混音后的音频数据流转发到 CDN 上。

步骤4：设置是否开启云端录制（可选）

- **OutputParams.RecordId**
  该参数用于指定是否启动 云端录制，如果您指定此参数，那么混流后的音视频流会被录制成文件并存储到 云点播 中。录制下来的文件会按照 `OutputParams.RecordId_开始时间_结束时间` 的格式命名，例如：`file001_2020-02-16-12-12-12_2020-02-16-13-13-13`。
- **OutputParams.RecordAudioOnly**
  如果您只希望录制音频而不需要视频内容，可以设置 `OutputParams.RecordAudioOnly` 参数为1，表示仅录制 MP3 格式的文件。

> **说明：**
> 自2022年08月01日起，新创建应用请参见 新版云端录制功能说明 发起录制，StartMCUMixTranscode 仅可用于 旧版云端录制 功能应用发起录制。如需确认当前应用的云端录制功能类型请参见 新、旧录制功能类型判断方式。

结束混流

由您的服务器调用 REST API StopMCUMixTranscode 即可结束混流。

方案二：客户端 SDK API 混流方案

使用 TRTC SDK 发起混流指令非常简单，只需调用各个平台的 setMixTranscodingConfig() API 即可，目前 SDK 提供了 4 种常用的混流方案：

参数项	纯音频模式（PureAudio）	预排版模式（PresetLayout）	屏幕分享模式（ScreenSharing）	全手动模式（Manual）
调用次数	只需调用一次接口	只需调用一次接口	只需调用一次接口	以下场景需调用混流接口： • 有连麦者加入时 • 有连麦者离开时 • 连麦者开关摄像头时 • 连麦者开关麦克风时
混合内容	只混合音频	自定义设置各路内容	不混合学生端的画面	自定义设置各路内容
audioSampleRate	推荐48000	推荐48000	推荐48000	推荐48000
audioBitrate	推荐64	推荐64	推荐64	推荐64
audioChannels	推荐2	推荐2	推荐2	推荐2
videoWidth	无需设置	不能为0	推荐0	不能为0
videoHeight	无需设置	不能为0	推荐0	不能为0
videoBitrate	无需设置	不能为0	推荐0	不能为0
videoFramerate	无需设置	推荐15	推荐15	推荐15
videoGOP	无需设置	推荐3	推荐3	推荐3
mixUsers 数组	无需设置	使用占位符设置	无需设置	使用真实 userId 设置

纯音频模式（PureAudio）

适用场景

纯音频模式适用于语音通话（AudioCall）和语音聊天室（VoiceChatRoom）等纯音频应用场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时进行设定。
纯音频模式下，SDK 会自动将房间里的多路音频流混合成一路。

使用步骤

1. 在调用 `enterRoom()` 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 `TRTCAppSceneAudioCall` 或 `TRTCAppSceneVoiceChatRoom`，明确当前房间中没有视频且只有音频。
2. 开启 旁路直播，并设定 TRTCParams 中的 `streamId` 参数，指定 MCU 输出的混合音频流的去处。
3. 调用 `startLocalAudio()` 开启本地音频采集和音频上行。

> **说明**
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音频上行才能构成混流的前提条件。

4. 调用 `setMixTranscodingConfig()` 接口启动云端混流，需要您在调用时将 `TRTCTranscodingConfig` 中的 `mode` 参数设定为 **TRTCTranscodingConfigMode_Template_PureAudio**，并指定 `audioSampleRate`、`audioBitrate` 和 `audioChannels` 等关乎音频输出质量的参数。
5. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> **注意**
> 纯音频模式下 `setMixTranscodingConfig()` 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。

预排版模式（PresetLayout）

适用场景

预排版模式适用于视频通话（VideoCall）和互动直播（LIVE）等音频和视频皆有的应用场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时进行设定。
预排版模式下，SDK 会自动按照您预先设定各路画面的排版规则将房间里的多路音频流混合成一路。

使用步骤

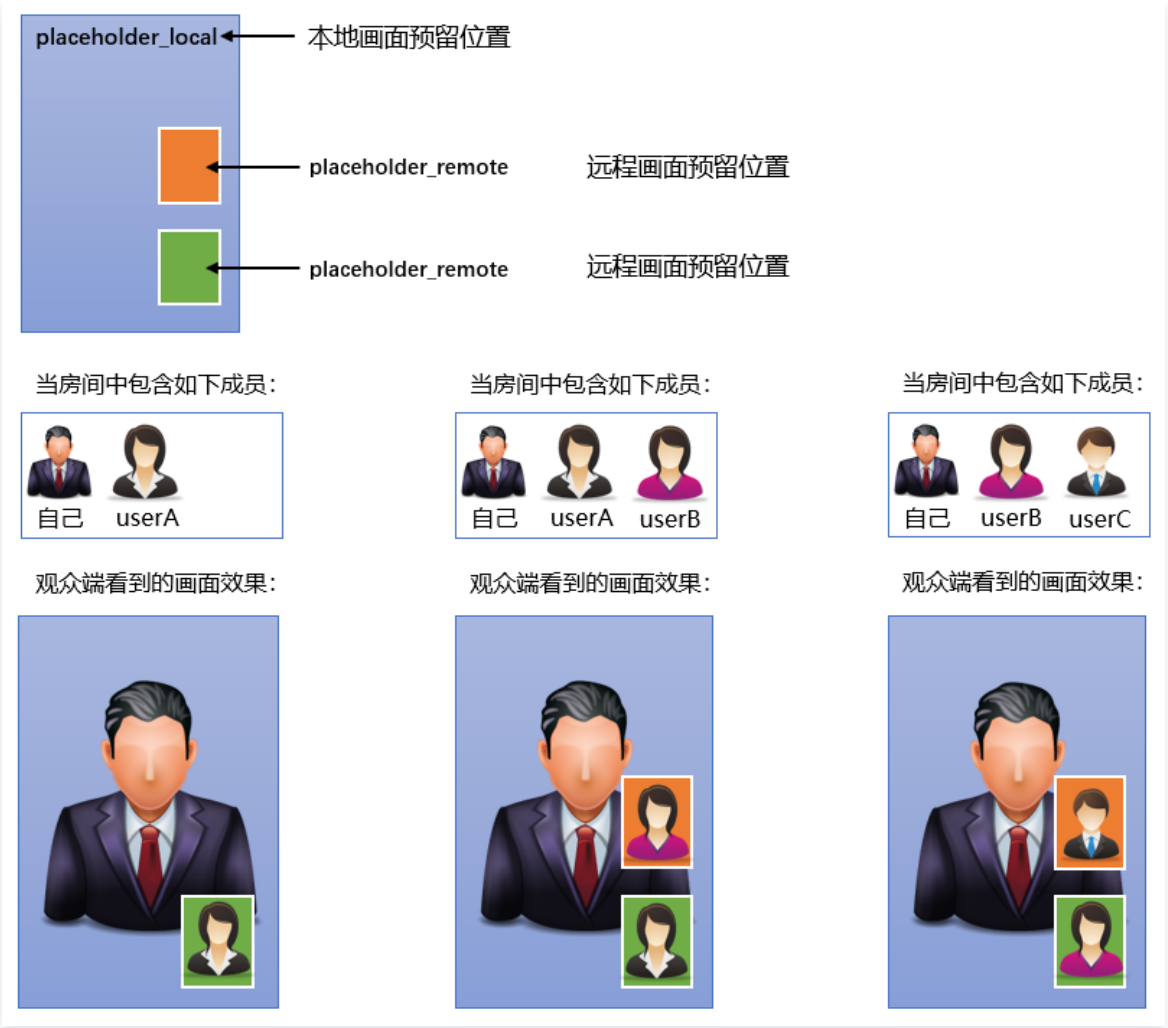
1. 在调用 `enterRoom()` 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 `TRTCAppSceneVideoCall` 或 `TRTCAppSceneLIVE`。
2. 开启 旁路直播，并设定 TRTCParams 中的 `streamId` 参数，指定 MCU 输出的混合音频流的去处。
3. 调用 `startLocalPreview()` 和 `startLocalAudio()` 开启本地的音视频上行。

> **说明**
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 `setMixTranscodingConfig()` 接口启动云端混流，需要您在调用时将 `TRTCTranscodingConfig` 中的 `mode` 参数设定为 **TRTCTranscodingConfigMode_Template_PresetLayout**，并指定 `audioSampleRate`、`audioBitrate` 和 `audioChannels` 等关乎音频输出质量的参数，以及 `videoWidth`、`videoHeight`、`videoBitrate`、`videoFramerate` 等关乎视频输出质量的参数。
5. 组装 `mixUser` 参数，预排版模式下 `mixUser` 中的 `userId` 参数请使用 **$PLACE_HOLDER_REMOTE$**、**$PLACE_HOLDER_LOCAL_MAIN$** 以及 **$PLACE_HOLDER_LOCAL_SUB$** 这三个占位字符串，其含义如下表所示：

占位符	含义	是否支持多个
$PLACE_HOLDER_LOCAL_MAIN$	指代本地摄像头这一路	不支持
$PLACE_HOLDER_LOCAL_SUB$	指代本地屏幕分享这一路（只有画面）	不支持
$PLACE_HOLDER_REMOTE$	指代远端连麦者，可以同时设置多个	支持

6. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。



示例代码

您可根据下面的示例代码实现“一大二小，上下叠加”的混合效果：

iOS

```
TRTCTranscodingConfig *config = [[TRTCTranscodingConfig alloc] init];
// 设置分辨率为720 × 1280，码率为1500kbps，帧率为20FPS
config.videoWidth       = 720;
config.videoHeight      = 1280;
config.videoBitrate     = 1500;
config.videoFramerate   = 20;
```

```
config.videoGOP         = 2;
config.audioSampleRate = 48000;
config.audioBitrate     = 64;
config.audioChannels    = 2;
// 采用预排版模式
config.mode = TRTCTranscodingConfigMode_Template_PresetLayout;

NSMutableArray *mixUsers = [NSMutableArray new];
// 主播摄像头的画面位置
TRTCMixUser* local = [TRTCMixUser new];
local.userId = @"$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0;    // zOrder 为0代表主播画面位于最底层
local.rect    = CGRectMake(0, 0, videoWidth, videoHeight);
local.roomID = nil; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:local];

// 连麦者的画面位置
TRTCMixUser* remote1 = [TRTCMixUser new];
remote1.userId = @"$PLACE_HOLDER_REMOTE$";
remote1.zOrder = 1;
remote1.rect    = CGRectMake(400, 800, 180, 240); //仅供参考
remote1.roomID = @"97392"; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:remote1];

// 连麦者的画面位置
TRTCMixUser* remote2 = [TRTCMixUser new];
remote2.userId = @"$PLACE_HOLDER_REMOTE$";
remote2.zOrder = 1;
remote2.rect    = CGRectMake(400, 500, 180, 240); //仅供参考
remote2.roomID = @"97392"; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:remote2];

config.mixUsers = mixUsers;
// 发起云端混流
[_trtc setMixTranscodingConfig:config];
```

Android

```
TRTCCloudDef.TRTCTranscodingConfig config = new TRTCCloudDef.TRTCTranscodingConfig();
// 设置分辨率为720 × 1280，码率为1500kbps，帧率为20FPS
config.videoWidth       = 720;
config.videoHeight      = 1280;
config.videoBitrate     = 1500;
config.videoFramerate   = 20;
config.videoGOP         = 2;
config.audioSampleRate = 48000;
config.audioBitrate     = 64;
config.audioChannels    = 2;

// 采用预排版模式
config.mode = TRTCCloudDef.TRTC_TranscodingConfigMode_Template_PresetLayout;
config.mixUsers = new ArrayList<>();

// 主播摄像头的画面位置
TRTCCloudDef.TRTCMixUser local = new TRTCCloudDef.TRTCMixUser();
```

```
local.userId = "$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0;    // zOrder 为0代表主播画面位于最底层
local.x      = 0;
local.y      = 0;
local.width  = videoWidth;
local.height = videoHeight;
local.roomId = null; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(local);

// 连麦者的画面位置
TRTCCloudDef.TRTCMixUser remote1 = new TRTCCloudDef.TRTCMixUser();
remote1.userId = "$PLACE_HOLDER_REMOTE$";
remote1.zOrder = 1;
remote1.x      = 400; //仅供参考
remote1.y      = 800; //仅供参考
remote1.width  = 180; //仅供参考
remote1.height = 240; //仅供参考
remote1.roomId = "97392"; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(remote1);

// 连麦者的画面位置
TRTCCloudDef.TRTCMixUser remote2 = new TRTCCloudDef.TRTCMixUser();
remote2.userId = "$PLACE_HOLDER_REMOTE$";
remote2.zOrder = 1;
remote1.x      = 400; //仅供参考
remote1.y      = 500; //仅供参考
remote1.width  = 180; //仅供参考
remote1.height = 240; //仅供参考
remote1.roomId = "97393"; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(remote2);

// 发起云端混流
trtc.setMixTranscodingConfig(config);
```

C++

```
TRTCTranscodingConfig config;
// 设置分辨率为1280 × 720，码率为1500kbps，帧率为20fps
config.videoWidth      = 1280;
config.videoHeight     = 720;
config.videoBitrate    = 1500;
config.videoFramerate  = 20;
config.videoGOP        = 2;
config.audioSampleRate = 48000;
config.audioBitrate    = 64;
config.audioChannels   = 2;

// 采用预排版模式
config.mode == TRTCTranscodingConfigMode_Template_PresetLayout
TRTCMixUser* mixUsersArray = new TRTCMixUser[3];
mixUsersArray[0].userId      = "$PLACE_HOLDER_LOCAL_MAIN$";
mixUsersArray[0].zOrder      = 0; // zOrder 为0代表主播画面位于最底层
mixUsersArray[0].rect.left   = 0;
mixUsersArray[0].rect.top    = 0;
```

```
mixUsersArray[0].rect.right  = videoWidth;
mixUsersArray[0].rect.bottom = videoHeight;
mixUsersArray[0].roomId      = nullptr; // 本地用户不用填写 roomID，远程需要

mixUsersArray[1].userId      = "$PLACE_HOLDER_REMOTE$";
mixUsersArray[1].zOrder      = 1;
mixUsersArray[1].rect.left   = 400; //仅供参考
mixUsersArray[1].rect.top    = 800; //仅供参考
mixUsersArray[1].rect.right  = 180; //仅供参考
mixUsersArray[1].rect.bottom = 240; //仅供参考
mixUsersArray[1].roomId      = "97392"; // 本地用户不用填写 roomID，远程需要

mixUsersArray[2].userId      = "$PLACE_HOLDER_REMOTE$";
mixUsersArray[2].zOrder      = 1;
mixUsersArray[2].rect.left   = 400; //仅供参考
mixUsersArray[2].rect.top    = 500; //仅供参考
mixUsersArray[2].rect.right  = 180; //仅供参考
mixUsersArray[2].rect.bottom = 240; //仅供参考
mixUsersArray[2].roomId      = "97393"; // 本地用户不用填写 roomID，远程需要
config.mixUsersArray = mixUsersArray;

// 发起云端混流
trtc->setMixTranscodingConfig(&config);
```

C#

```
TRTCTranscodingConfig config = new TRTCTranscodingConfig();
// 设置分辨率为1280 × 720，码率为1500kbps，帧率为20fps
config.videoWidth      = 1280;
config.videoHeight     = 720;
config.videoBitrate    = 1500;
config.videoFramerate  = 20;
config.videoGOP        = 2;
config.audioSampleRate = 48000;
config.audioBitrate    = 64;
config.audioChannels   = 2;
config.mode = RTCTranscodingConfigMode.TRTCTranscodingConfigMode_Template_PresetLayout;
TRTCMixUser[] mixUsersArray = new TRTCMixUser[3];

// 主播摄像头的画面位置
TRTCMixUser local = new TRTCMixUser();
local.userId = "$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0; // zOrder 为0代表主播画面位于最底层
local.roomId = null; // 本地用户不用填写 roomID，远程需要
RECT rtLocal = new RECT() {
    left   = 0,
    top    = 0,
    right  = videoWidth,
    bottom = videoHeight
};
local.rect = rtLocal;
mixUsersArray[0] = local;

// 连麦者的画面位置
```

```
TRTCMixUser remote1 = new TRTCMixUser();
remote1.userId = "$PLACE_HOLDER_REMOTE$";
remote1.zOrder = 1;
remote1.roomId = “97392”; // 本地用户不用填写 roomID，远程需要
RECT rtRemote1 = new RECT() { //仅供参考
    left   = 420,
    top    = 81,
    right  = 240 + left,
    bottom = 240 + top
};
remote1.rect = rtRemote1;
mixUsersArray[1] = remote1;

// 连麦者的画面位置
TRTCMixUser remote2 = new TRTCMixUser();
remote2.userId = "$PLACE_HOLDER_REMOTE$";
remote2.zOrder = 1;
remote2.roomId = “97393”; // 本地用户不用填写 roomID，远程需要
RECT rtRemote2 = new RECT() { //仅供参考
    left   = 660,
    top    = 400,
    right  = 240 + left,
    bottom = 240 + top
};
rtRemote2.rect   = rtRemote2;
mixUsersArray[2] = remote2;

// 发起云端混流
config.mixUsersArray = mixUsersArray;
trtc.setMixTranscodingConfig(config);
```

Flutter

```
TRTCCloud trtcCloud = await TRTCCloud.sharedInstance();
trtcCloud.setMixTranscodingConfig(TRTCTranscodingConfig(
  appId: 1252463788, //仅供参考
  bizId: 3891, //仅供参考
  // 设置分辨率为720 × 1280，码率为1500kbps，帧率为20FPS
  videoWidth: 720,
  videoHeight: 1280,
  videoBitrate: 1500,
  videoFramerate: 20,
  videoGOP: 2,
  audioSampleRate: 48000,
  audioBitrate: 64,
  audioChannels: 2,

  // 采用预排版模式
  mode: TRTCCloudDef.TRTC_TranscodingConfigMode_Template_PresetLayout,

  mixUsers: [
  // 主播摄像头的画面位置
    TRTCMixUser(
      userId: "$PLACE_HOLDER_LOCAL_MAIN$",
```

```
      roomId: null, // 本地用户不用填写 roomID，远程需要
      zOrder: 0, // zOrder 为0代表主播画面位于最底层
      x: 0, //仅供参考
      y: 0,
      streamType: 0,
      width: 300,
      height: 400),
    TRTCMixUser(
      userId: '$PLACE_HOLDER_REMOTE$',
      roomId: '256', // 本地用户不用填写 roomID，远程需要
      zOrder: 1,
      x: 100, //仅供参考
      y: 100,
      streamType: 0,
      width: 160,
      height: 200)
  ],
));
```

Web

```
try {
  const config = {
    target: {
      publishMode: PublishMode.publishMixStreamToCDN
    },
    encoding: {
      videoWidth: 720,
      videoHeight: 1280,
      videoBitrate: 1500,
      videoFramerate: 20,
      videoGOP: 2,
      audioSampleRate: 48000,
      audioBitrate: 64,
      audioChannels: 2,
    },
    // 一路本地摄像头、两路远端流的排版位置
    mix: {
      audioMixUserList: [
        {
          userId: 'jack'
        },
        {
          userId: 'jack2'
        },
        {
          userId: 'jack3'
        }
      ],
      videoLayoutList: [
        {
          width: 720,
          height: 1280,
          locationX: 0,
```

```
          locationY: 0,
          userId: 'jack', // 本地摄像头占位，传入推摄像头的 client userId
          zOrder: 1
        },
        {
          width: 180,
          height: 240,
          locationX: 400,
          locationY: 800,
          userId: 'jack2', //  Web 端目前不支持预排版模式，需要传入实际的 userId
          zOrder: 2
        },
        {
          width: 180,
          height: 240,
          locationX: 400,
          locationY: 500,
          userId: 'jack3',
          zOrder: 2
        }
      ]
    }
  }
  await trtc.startPlugin('CDNStreaming', config);
} catch (error) {
  console.error('CDNStreaming start failed ', error);
}
```

注意

- 预排版模式下 `setMixTranscodingConfig()` 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。
- 上述 Web 端是 5.x 版本的代码示例，详细教程请参考 CDNStreaming Plugin 。
- 4.x 版本请参考： Client.startMixTranscode() 。

屏幕分享模式（ScreenSharing）

适用场景

屏幕分享模式适用于在线教育和互动课堂等场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时将 AppScene 参数设定为 `TRTCAppSceneLIVE` 。
屏幕分享模式下，SDK 会先根据您所选定的目标分辨率构建一张画布。当老师未开启屏幕分享时，SDK 会将摄像头画面等比例拉伸绘制到该画布上；当老师开启屏幕分享后，SDK 会将屏幕分享画面绘制到同样的画布上。通过构建画布可以确保混流模块的输出分辨率一致，防止录制和网页观看的视频兼容性问题（普通播放器不支持分辨率会变化的视频）。

使用步骤

1. 在调用 `enterRoom()` 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 `TRTCAppSceneLIVE` 。
2. 开启 旁路直播，并设定 TRTCParams 中的 `streamId` 参数，指定 MCU 输出的混合音视频流的去处。
3. 调用 `startLocalPreview()` 和 `startLocalAudio()` 开启本地的音视频上行。

说明

> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 `setMixTranscodingConfig()` 接口启动云端混流，需要您在调用时将 `TRTCTranscodingConfig` 中的 `mode` 参数设定为 **TRTCTranscodingConfigMode_Template_ScreenSharing**，并指定 `audioSampleRate`、`audioBitrate` 和 `audioChannels` 等关乎音频输出质量的参数，以及 `videoWidth`、`videoHeight`、`videoBitrate`、`videoFramerate` 等关乎视频输出质量的参数。

> **说明**
> 若将 `videoWidth` 和 `videoHeight` 参数均指定为0，SDK 会自动根据用户当前屏幕的宽高比计算出一个合适的分辨率。

5. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> **注意**
> - 屏幕分享模式下 `setMixTranscodingConfig()` 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。
> - 由于教学模式下的视频内容以屏幕分享为主，同时传输摄像头画面和屏幕分享画面非常浪费带宽。建议直接将摄像头画面和学生的画面通过 `setLocalVideoRenderCallback()` 和 `setRemoteVideoRenderCallback()` 接口自绘到当前屏幕上。
> - 通过将 TRTCTranscodingConfig 中的 `videoWidth` 和 `videoHeight` 参数均指定为 0，可以让 SDK 智能选择输出分辨率。如果老师当前屏幕宽度小于1920px，SDK 会使用老师当前屏幕的实际分辨率；如果老师当前屏幕宽度大于1920px，SDK 会根据当前屏幕宽高比，选择 1920 × 1080（16:9）、1920 × 1200（16:10）或1920 × 1440（4:3）。

全手动模式（Manual）

适用场景

全手动模式适合于上述自动模式均不适用的场景，全手动的灵活性最高，可以自由组合出各种混流方案，但易用性最差。
全手动模式下，您需要设置 `TRTCTranscodingConfig` 中的所有参数，并需要监听 TRTCCloudDelegate 中的 `onUserVideoAvailable()` 和 `onUserAudioAvailable()` 回调，以便根据当前房间中各个上麦用户的音视频状态不断地调整 `mixUsers` 参数，否则会导致混流失败。

使用步骤

1. 在调用 `enterRoom()` 函数进入房间时，根据您的业务需要，设定 AppScene 参数。
2. 开启 旁路直播，并设定 TRTCParams 中的 `streamId` 参数，指定 MCU 输出的混合音视频流的去处。
3. 根据您的业务需要，调用 `startLocalAudio()` 开启本地的音频上行（或同时调用 `startLocalPreview()` 开启视频上行）。

> **说明**
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 `setMixTranscodingConfig()` 接口启动云端混流，需要您在调用时将 `TRTCTranscodingConfig` 中的 `mode` 参数设定为 **TRTCTranscodingConfigMode_Manual**，并指定 `audioSampleRate`、`audioBitrate` 和 `audioChannels` 等关乎音频输出质量的参数。如果您的业务场景中也包含视频，需同时设置 `videoWidth`、`videoHeight`、`videoBitrate`、`videoFramerate` 等关乎视频输出质量的参数。
5. 监听 TRTCCloudDelegate 中的 `onUserVideoAvailable()` 和 `onUserAudioAvailable()` 回调，并根据需要指定**mixUsers**参数。

> **说明**
> 与预排版（PresetLayout）模式不同，Manual 需要您指定每一个 `mixUser` 中的 `userId` 参数为真实的连麦者 ID，并且也要根据该连麦者是否开启了视频，如实设定 `mixUser` 中的 `pureAudio` 参数。

6. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> **注意**
> 全手动模式下，您需要实时监听房间中连麦者的上麦下麦动作，并根据连麦者的人数和音视频状态，多次调用 `setMixTranscodingConfig()` 接口。

相关费用

费用的计算

云端混流转码需要对输入 MCU 集群的音视频流进行解码后重新编码输出，将产生额外的服务成本，因此 TRTC 将向使用 MCU 集群进行云端混流转码的用户收取额外的增值费用。云端混流转码费用根据**转码输出的分辨率大小**和**转码时长**进行计费，转码输出的分辨率越高、转码输出的时间越长，费用越高。详情请参见 云端混流转码计费说明 。

费用的节约

- 在**基于服务端 REST API 混流方案下，要停止混流**，需要满足如下条件之一：
  - 房间里的所有用户（包括主播和观众）都退出了房间。
  - 调用 REST API StopMCUMixTranscode 主动停止混流。
- 在**基于客户端 SDK API 混流方案下，要停止混流**，需要满足如下条件之一：
  - 发起混流（即调用了客户端 API `setMixTranscodingConfig`）的主播退出了房间。
  - 调用 setMixTranscodingConfig 并将参数 config 设置为 `nil/null` 主动停止混流。

在其他情况下，TRTC 云端都将会尽力持续保持混流状态。因此，为避免产生预期之外的混流费用，请在您不需要混流的时候尽早通过上述方法结束云端混流。

CDN直播观看(旧)

最近更新时间：2024-12-03 16:33:13

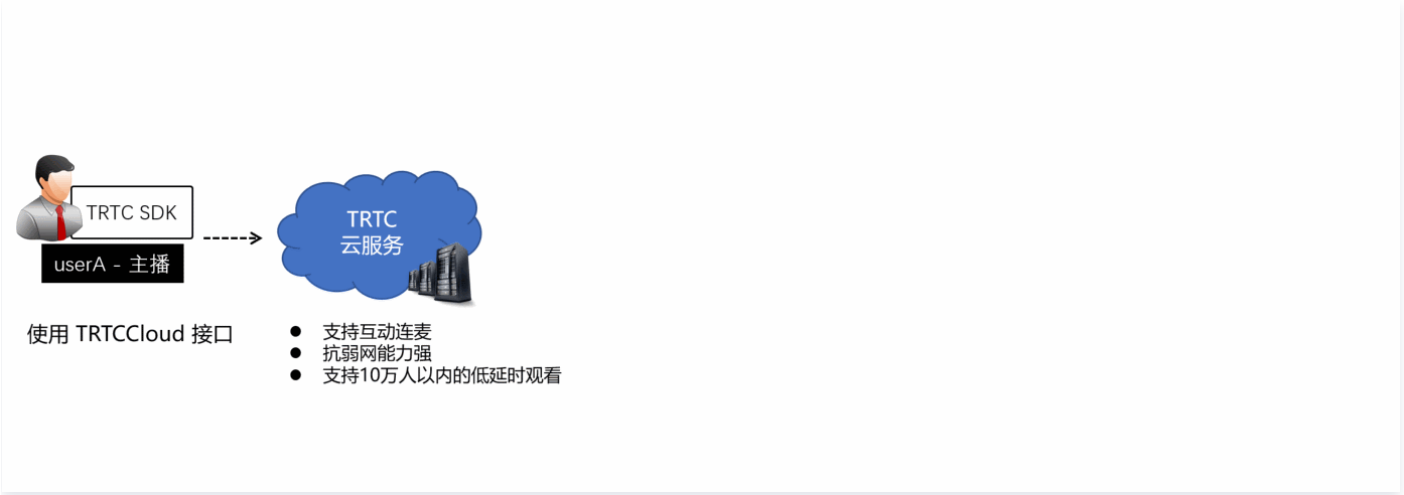
适用场景

CDN 直播观看，也叫 “CDN 旁路直播”，由于 TRTC 采用 UDP 协议进行传输音视频数据，而标准直播 CDN 则采用的 RTMP\HLS\FLV 等协议进行数据传输，所以需要将 TRTC 中的音视频数据**旁路**到直播 CDN 中，才能在让观众通过直播 CDN 进行观看。
给 TRTC 对接 CDN 观看，一般被用于解决如下两类问题：

- **问题一：超高并发观看**
  TRTC 的低延时观看能力，单房间支持的最大人数上限为10万人。CDN 观看虽然延迟要高一些，但支持10万人以上的并发观看，且 CDN 的计费价格更加便宜。
- **问题二：移动端网页播放**
  TRTC 虽然支持 WebRTC 协议接入，但主要用于 Chrome 桌面版浏览器，移动端浏览器的兼容性非常不理想，尤其是 Android 手机浏览器对 WebRTC 的支持普遍都很差。所以如果希望通过 Web 页面在移动端分享直播内容，还是推荐使用 HLS(m3u8) 播放协议，这也就需要借助直播 CDN 的能力来支持 HLS 协议。

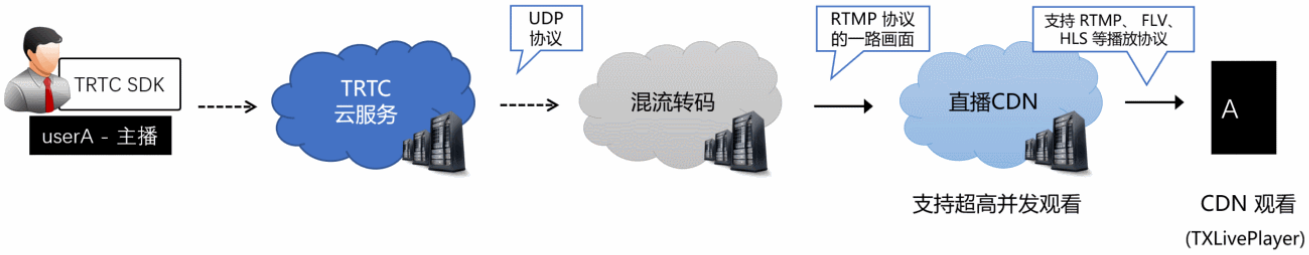
原理解析

腾讯云会使用一批旁路转码集群，将 TRTC 中的音视频数据旁路到直播 CDN 系统中，该集群负责将 TRTC 所使用的 UDP 协议转换为标准的直播 RTMP 协议。
单路画面的旁路直播
当 TRTC 房间中只有一个主播时，TRTC 的旁路推流跟标准的 RTMP 协议直推功能相同，不过 TRTC 的 UDP 相比于 RTMP 有更强大的弱网络抗性。



混合画面的旁路直播
TRTC 最擅长的领域就是音视频互动连麦，如果一个房间里同时有多个主播，而 CDN 观看端只希望拉取一路音视频画面，就需要使用 云端混流服

务 将多路画面合并成一路，其原理如下图所示：



说明

为什么不直接播放多路 CDN 画面?

播放多路 CDN 画面很难解决多路画面的延迟对齐问题，同时拉取多路画面所消耗的下载流量也比单独画面要多，所以业内普遍采用云端混流方案。

前提条件

已开通腾讯 云直播 服务。应国家相关部门的要求，直播播放必须配置播放域名，具体操作请参考 添加自有域名 。

使用步骤

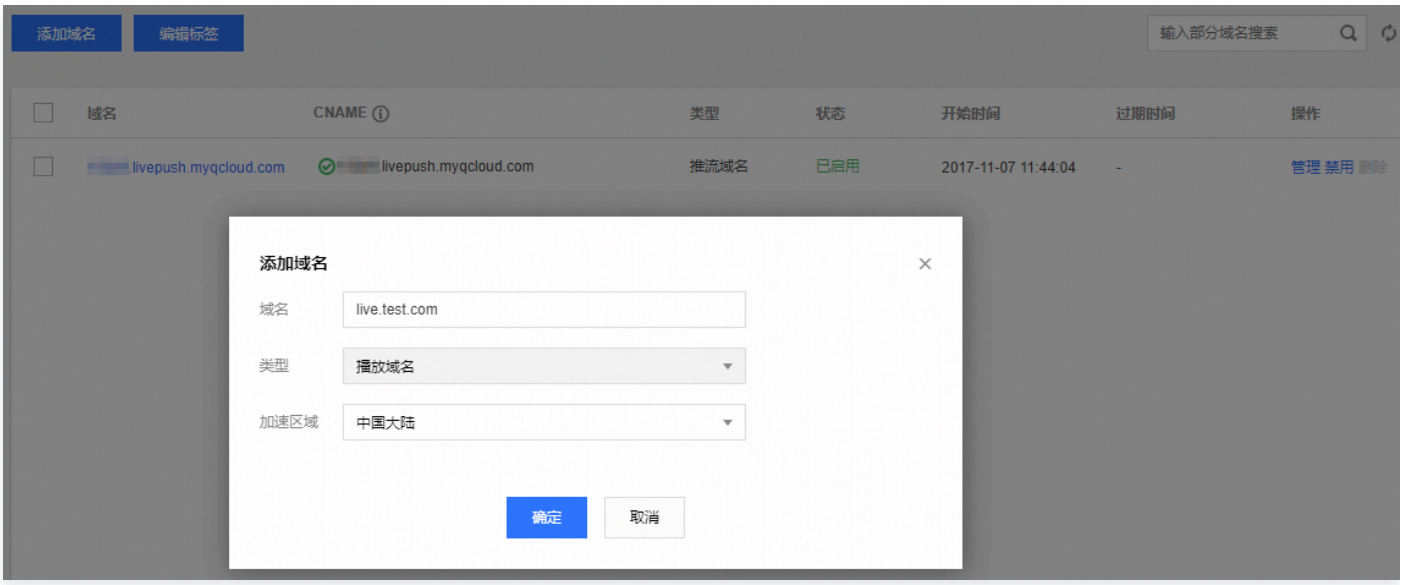
步骤1：开启旁路推流功能

1. 登录 实时音视频控制台 。
2. 在左侧导航栏选择**应用管理**，单击目标应用所在行的**功能配置**。
3. 在**旁路推流配置**中，单击**启用旁路推流**右侧的开关按钮，在弹出的**开启旁路推流功能**对话框中，单击**开启旁路推流功能**即可开通。

步骤2：配置播放域名并完成 CNAME

1. 登录 云直播控制台 。
2. 在左侧导航栏选择**域名管理**，您会看到在您的域名列表新增了一个推流域名，格式为 `xxxxx.livepush.myqcloud.com` ，其中 xxxxx 是一个数字，叫做 bizid，您可以在**实时音视频控制台** > 应用管理 > 详情 >功能配置（基础）中旁路转推配置下的默认推流域名查找到 bizid 信息。

3. 单击**添加域名**，输入您已经备案过的播放域名，选择域名类型为**播放域名**，选择加速区域（默认为**中国大陆**），单击**确定**即可。
4. 域名添加成功后，系统会为您自动分配一个 CNAME 域名（以 `.liveplay.myqcloud.com` 为后缀）。CNAME 域名不能直接访问，您需要在域名服务提供商处完成 CNAME 配置，配置生效后，即可享受云直播服务。具体操作请参见 CNAME 配置 。



> **注意**
> 在 步骤1 中开启旁路直播功能后，腾讯云会默认在您的云直播控制台中增加一个格式为 `xxxxx.livepush.myqcloud.com` 的推流域名，该域名为腾讯云直播服务和 TRTC 服务之间约定的一个默认推流域名。

步骤3：关联 TRTC 的音视频流到直播 streamId

开启旁路推流功能后， TRTC 房间里的每一路画面都配备一路对应的播放地址，该地址的格式如下：

```
http://播放域名/live/[streamId].flv
```

地址中的 streamId 可以在直播中唯一标识一条直播流，您可以自己指定 streamId，也可以使用系统默认生成的。

方式一：自定义指定 streamId

您可以在调用 `TRTCCloud` 的 `enterRoom` 函数时，通过其参数 `TRTCParams` 中的 `streamId` 参数指定直播流 ID。
以 iOS 端的 Objective-C 代码为例：

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
TRTCParams *param = [[TRTCParams alloc] init];
param.sdkAppId = 1400000123;      // TRTC 的 SDKAppID，创建应用后可获得
param.roomId   = 1001;            // 房间号
param.userId   = @"rexchang";     // 用户名
param.userSig  = @"xxxxxxxx";     // 登录签名
param.role     = TRTCRoleAnchor; // 角色：主播
param.streamId = @"stream1001";   // 流 ID
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE]; // 请使用 LIVE 模式
```

userSig 的计算方法请参见 如何计算及使用 UserSig 。

方式二：系统指定 streamId

开启自动旁路推流后，如果您没有自定义指定 streamId，系统会默认为您生成一个缺省的 streamId，生成规则如下：

- **拼装 streamId 用到的字段**
  - SDKAppID：您可以在 控制台 > 应用管理 > 应用信息中查找到。
  - bizid：您可以在 控制台 > 应用管理 > 应用信息中查找到。
  - roomId：由您在 `enterRoom` 函数的参数 `TRTCParams` 中指定。

- userId：由您在 `enterRoom` 函数的参数 `TRTCParams` 中指定。
- streamType：摄像头画面为 main，屏幕分享为 aux（WebRTC 由于同时只支持一路上行，因此 WebRTC 上屏幕分享的流类型是 main）。

- **拼装 streamId 的计算规则**

拼装	2020年01月09日及此后新建的应用	2020年01月09日前创建且使用过的应用
拼装规则	streamId = urlencode(sdkAppId_roomId_userId_streamType)	StreamId = bizid_MD5(roomId_userId_streamType)
计算样例	例如：sdkAppId = 12345678，roomId = 12345，userId = userA，用户当前使用了摄像头。那么：streamId = 12345678_12345_userA_main	例如：bizid = 1234，roomId = 12345，userId = userA，用户当前使用了摄像头。那么：streamId = 1234_MD5(12345_userA_main) = 1234_8D0261436C375BB0DEA901D86D7D70E8

步骤4：控制多路画面的混合方案

如果您想要获得混合后的直播画面，需要调用 TRTCCloud 的 `setMixTranscodingConfig` 接口启动云端混流转码，该接口的参数 `TRTCTranscodingConfig` 可用于配置：

- 各个子画面的摆放位置和大小。
- 混合画面的画面质量和编码参数。

画面布局的详细配置方法请参考 云端混流转码 。

> **注意**
> `setMixTranscodingConfig` 并不是在终端进行混流，而是将混流配置发送到云端，并在云端服务器进行混流和转码。由于混流和转码都需要对原来的音视频数据进行解码和二次编码，所以需要更长的处理时间。因此，混合画面的实际观看时延要比独立画面的多出1s - 2s。

步骤5：给 SDK 配置 License 授权

实时音视频（TRTC）SDK 提供了功能全面性能强大的直播播放能力，可轻松配合云直播实现 CDN 直播观看功能。若您使用移动端（iOS&Android）10.1及其之后版本的实时音视频（TRTC）SDK 实现 CDN 直播观看，则须配置 License 授权，否则可跳过本步骤。

1. 获取 License 授权：
   - 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。

   - 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。
2. 在您的 App 调用 SDK 相关功能之前（建议在 `Application` / `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

Android

```
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
    V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
            }
        });
}
```

iOS

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

注意

License 中配置的 packageName/BundleId 必须和应用本身一致，否则会播放失败。

步骤6：获取播放地址并对接播放

当您通过 步骤2 配置完播放域名和 步骤3 完成 streamId 的映射后，即可得到直播的播放地址。播放地址的标准格式为：

```
http://播放域名/live/[streamId].flv
```

例如，您的播放域名为 `live.myhost.com`，您将房间（1001）中的用户 userA 的直播流 ID 通过进房参数指定为 streamId = "streamd1001"。
则您可以得到三路播放地址：

```
rtmp 协议的播放地址：rtmp://live.myhost.com/live/streamd1001
flv 协议的播放地址：http://live.myhost.com/live/streamd1001.flv
hls 协议的播放地址：http://live.myhost.com/live/streamd1001.m3u8
```

我们推荐以 `http` 为前缀且以 `.flv` 为后缀的 **http - flv** 地址，该地址的播放具有时延低、秒开效果好且稳定可靠的特点。
播放器选择方面推荐参考如下表格中的指引的方案：

所属平台	对接文档	API 概览	支持的格式
iOS App	接入指引	V2TXLivePlayer(iOS)	推荐 FLV
Android App	接入指引	V2TXLivePlayer(Android)	推荐 FLV
Web 浏览器	接入指引	-	• 桌面端 Chrome 浏览器支持 FLV • Mac 端 Safari 和移动端手机浏览器仅支持 HLS
微信小程序	接入指引	<live-player> 标签	推荐 FLV

步骤7：优化播放延时

开启旁路直播后的 http - flv 地址，由于经过了直播 CDN 的扩散和分发，观看时延肯定要比直接在 TRTC 直播间里的通话时延要高。
按照目前腾讯云的直播 CDN 技术，如果配合 V2TXLivePlayer 播放器，可以达到下表中的延时标准：

旁路流类型	V2TXLivePlayer 的播放模式	平均延时	实测效果
独立画面	极速模式（推荐）	2s - 3s	下图中左侧对比图（橙色）
混合画面	极速模式（推荐）	4s - 5s	下图中右侧对比图（蓝色）

下图中的实测效果，采用了同样的一组手机，左侧 iPhone 6s 使用了 TRTC SDK 进行直播，右侧的小米6 使用 V2TXLivePlayer 播放器播放 FLV 协议的直播流。

如果您在实测中延时比上表中的更大，可以按照如下指引优化延时：

- **使用 TRTC SDK 自带的 V2TXLivePlayer**
  普通的 ijkplayer 或者 ffmpeg 基于 ffmpeg 的内核包装出的播放器，缺乏延时调控的能力，如果使用该类播放器播放上述直播流地址，时延一般不可控。V2TXLivePlayer 有一个自研的播放引擎，具备延时调控的能力。
- **设置 V2TXLivePlayer 的播放模式为极速模式**
  可以通过设置 V2TXLivePlayer 的参数来实现极速模式，以 iOS 为例：

```
//自动模式
[_txLivePlayer setCacheParams:1 maxTime:5];
//极速模式
[_txLivePlayer setCacheParams:1 maxTime:1];
//流畅模式
[_txLivePlayer setCacheParams:5 maxTime:5];

//设置完成之后再启动播放
```

相关费用

使用 CDN 直播观看需要云直播服务资源和终端 SDK 直播播放能力的配合，可能会产生以下费用。

云服务消耗

CDN 直播观看需要使用**云直播**的资源进行直播分发。云直播的费用主要包括**基础服务费用和增值服务费用**：基础服务主要是直播推流/播放产生的消耗；增值服务是直播过程中，使用增值服务产生的消耗。

> ⚠ **注意**
> 本文中的价格为示例，仅供参考。最终价格与计费策略请以 云直播 的计费说明为准。

- **基础服务费用：**
  将 TRTC 的内容旁路到云直播 CDN 观看时，**云直播**会收取观众观看产生的下行流量/带宽费用，可以根据实际需要选择适合自己的计费方式，默认采用流量计费，详情请参见 云直播 > 标准直播 > 流量带宽 计费说明。
- **增值服务费用：**
  如果您使用了云直播的转码、录制、云导播等功能，会产生对应额外的增值服务费用。增值服务可按需使用进行付费。

SDK 播放授权

实时音视频（TRTC）SDK 提供了功能全面性能强大的直播播放能力，可轻松配合云直播实现 CDN 直播观看功能。移动端 SDK 在10.1及以上的版本可通过获取指定 License 以解锁直播播放能力。

> ⚠ **注意**
> TRTC 的播放能力无需 License 授权。

您可直接 购买视频播放 License，或通过 购买的云点播流量包 免费获赠视频播放 License 或 短视频 License，两种 License 均可用于解锁 SDK 的视频播放功能。并且点播资源包可以抵扣云点播的播放产生的日结流量，详细说明请参见 云点播预付费资源包 。
License 计费说明参见 腾讯云视立方 License，License 购买完成后可参考 License 操作指引 进行新增和续期等操作。

常见问题

为什么房间里只有一个人时画面又卡又模糊？
请将 `enterRoom` 中 TRTCAppScene 参数指定为 **TRTCAppSceneLIVE**。
VideoCall 模式针对视频通话做了优化，所以在房间中只有一个用户时，画面会保持较低的码率和帧率以节省用户的网络流量，看起来会感觉又卡又模糊。

云端录制计费说明

最近更新时间：2024-08-23 14:12:22

> **注意：**
> - 自2020年7月1日起首次在 TRTC 控制台创建应用的腾讯云账号，使用云端录制功能后产生的录制费用以本文档的计费规则为准。
> - 在2020年7月1日之前已经在 TRTC 控制台创建过应用的腾讯云账号，无论是在2020年7月1日之前还是之后创建的应用，使用云端录制功能后产生的录制费用均默认继续沿用 云直播 > 直播录制 的计费规则。
> - 本文档仅针对 TRTC 云端录制的录制费用作出相关说明。云端录制完成后输出的录制文件默认保存在云点播平台，云点播将根据您的使用情况收取**存储费用**和**观看费用**，详情请参见 云端录制 > 相关费用 。
> - 如果您在云端录制之前使用了 云端混流转码 功能，还会产生额外的旁路转码费用，详情请参见 云端混流转码计费说明 。

用量统计方式

实时音视频 TRTC 按同一腾讯云账号下所有应用使用云端录制后输出结果中的**录制时长**来统计云端录制服务的用量。录制时长根据云端录制结果的不同，分为**视频时长**和**语音时长**。

> **注意：**
> 时长统计精度为秒，按 SDKAppID 维度，以每日累计秒数转换成分钟数后进行计费，不足一分钟按一分钟计。

视频时长

视频时长是指录制结果中包含视频画面的时间。TRTC 会根据录制的视频分辨率划分视频档位，然后分别对不同档位的视频时长进行计费。视频档位与分辨率的对应关系如下表所示：

视频档位	分辨率
标清 SD	不高于640 × 480（含）
高清 HD	640 × 480 – 1280 × 720（含）
全高清 FHD	高于720

- 同一个录制文件同一时间内，既有视频又有音频时，只按视频时长统计，不会重复计算语音时长。
- 录制实际输出分辨率可能会因为输入分辨率变化而变化，TRTC 将分段统计服务用量，每5秒更新一次。

语音时长

语音时长是指录制结果中只有纯音频的时间。

服务定价

TRTC 云端录制服务的刊例价如下表所示：

计费类型	单价（元/千分钟）
语音	3.50
标清 SD	7.00
高清 HD	14.00
全高清 FHD	52.50

计费方式

即支付方式。TRTC 云端录制服务仅支持**日结后付费**的方式，按日计费，每天上午10点扣除前一天产生的费用。

> ⚠ **注意：**
> 云端录制依赖**云直播**和**云点播**，如果腾讯云账户因为云直播或云点播欠费，将导致云端录制失败。

计费示例

> ⚠ **注意：**
> - 本文计费示例采用刊例价计算，如果您与腾讯云的商务经理签订了合同，以合同约定的价格为准。
> - 默认情况下，每个 TRTC 房间内每个用户的音视频流将分别录制成独立的文件，如果您希望将单个房间内多个用户的视频画面录制在一个文件内，可以通过 云端混流转码 将多个视频画面混合成1个。使用云端混流转码功能将产生额外的旁路转码费用，详情请参见 云端混流转码计费说明 。

假设用户 A、B、C 一起在 TRTC 房间中持续通话10分钟，需将整个通话过程完整录制下来。A、B、C 上行推流类型及分辨率信息如下表所示：

用户	推流类型	分辨率	对应计费类型
A	仅音频	无	语音
B	视频+音频	640 × 360	标清 SD
C	仅视频	1280 × 720	高清 HD

不混流直接录制示例

不混流的情况下，A、B、C 分别录制成独立的1个文件，3人共生成3个文件，则本次通话产生的录制费用为：

> A 的录制费用 + B 的录制费用 + C 的录制费用 = 语音时长单价 x A 的语音时长 + 标清视频时长单价 × B 的标清视频时长 + 高清视频时长单价 × C 的高清视频时长 = 3.5元/千分钟 x 10分钟 / 1000 + 7元/千分钟 × 10分钟 / 1000 + 14元/千分钟 × 10分钟 / 1000 = 0.245元。

混流后再录制示例

混流的情况下，将 A、B、C 混流后录制成1个文件，混合后输出的视频分辨率始终保持在1280 × 720，则本次通话产生的录制费用为：

> 高清视频时长单价 × 高清视频时长 = 14元/千分钟 × 10分钟 / 1000 = 0.14元。

使用云端混流转码功能将产生额外的旁路转码费用，详情请参见 云端混流转码计费说明 。

相关文档

- 免费试用
- 音视频时长计费说明
- 预付费套餐包
- 云端混流转码计费说明
- 开启后付费
- 折扣活动
- 退费说明
- 欠费停服说明
- 计费常见问题

开启后付费

最近更新时间：2024-09-20 21:28:32

本文主要讲述实时音视频增值服务和基础服务如何开启后付费。

基础服务

基础服务 根据具体应用场景可细分为语音互动直播、视频互动直播、语音通话和视频通话。您可根据实际业务需求开启后付费。

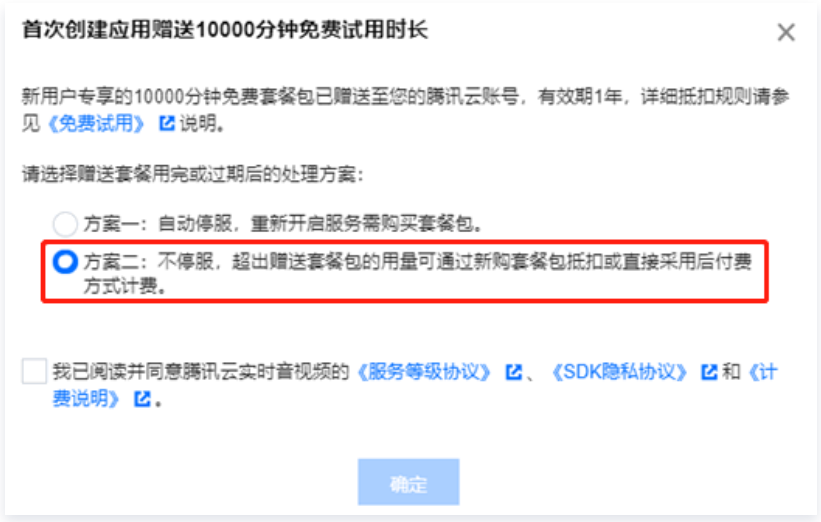
方案一： 控制台自助开启后付费

首次创建应用选择开启后付费

1. 首次开通 实时音视频 服务，选择**开发辅助** > **快速跑通Demo** 创建您的首个实时音视频应用。

2. 输入应用名称，例如 `TestTRTC` 。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

4. 您可通过赠送套餐包弹框选择“**不停服，超出赠送套餐包的用量可通过新购套餐包抵扣或直接采用后付费方式计费**”。



应用信息列表开启后付费

1. 若您首次创建应用时未开启后付费，后因实际业务要求赠送套餐包用完或过期后不停服，您可进入实时音视频控制台前往 应用管理 选择**应用信息**查看应用列表。
2. 单击其右侧操作栏的**应用信息**。

3. 进入应用详情页，通过**应用信息**页签的 “应用信息” 模块查看当前应用的基本信息。

4. 查看 “实时音视频服务状态” 单击**开通后付费**。

方案二：系统自动开启后付费

若您已购买通用套餐包，待套餐包都消耗完或过期后系统会**自动开启**，超出套餐包的用量将采用**后付费**的计费方式。

> **注意：**
> 当实时音视频基础服务用量无套餐包可抵扣或超出套餐包余量时，将采用后付费的方式，按**刊例价**计费。**后付费属于保底支付方式，开启后无法关闭**。

增值服务

增值服务指的是基于四种基础服务之上额外提供的增值功能，无法脱离基础服务单独使用，使用增值服务需支付额外的增值费用，TRTC 提供 云端录制 和 云端混流转码 两种增值服务。TRTC 增值服务仅支持**日结后付费**的方式，您无需自助开启后付费。

uni-app(小程序)快速接入(旧)

最近更新时间：2024-04-09 17:32:41

本文将介绍如何快速完成 TUICallKit 组件的接入，跟随本文档，您将在半小时内得到一个包含完备 UI 界面的视频通话小程序。基本功能如下图所示：

小程序 Demo 体验

- 如果您想要直接体验音视频通话小程序，单击 Demo 体验，扫描小程序二维码。
- 如果您想要**亲自集成** TUICallKit 组件，搭建一个音视频通话小程序，请跟随本文档。

开发环境要求

- 微信 App iOS 最低版本要求：7.0.9。
- 微信 App Android 最低版本要求：7.0.8。
- 小程序基础库最低版本要求：2.10.0。

警告：

- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，请使用**企业小程序账号**申请相关权限进行开发。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在**真机**上进行运行体验。

小程序开发准备

步骤一：开通企业类小程序

小程序推拉流标签不支持个人小程序，只支持企业类小程序。需要在 注册 时填写**主体类型为企业**，如下图所示：

步骤二：在小程序控制台开启实时音视频接口

- 小程序推拉流标签使用权限暂时只开放给有限类目，具体支持类目参考该地址 。
- 符合类目要求的小程序，需要在 微信公众平台 > 开发 > 开发管理 > 接口设置中自助开通该组件权限。

TUICallKit 源码集成

步骤一：创建 uni-app 小程序项目

1. 在 HBuilder 中创建小程序项目。

2. 在终端输入 `npm init -y`，创建 `package.json` 文件。

```
npm init -y
```

步骤二：下载并导入 TUICallKit 组件

MacOS 端

```
npm i @tencentcloud/call-uikit-wechat@1.4.4
```

```
mkdir -p ./wxcomponents/TUICallKit && cp -r node_modules/@tencentcloud/call-uikit-wechat/
./wxcomponents/TUICallKit
```

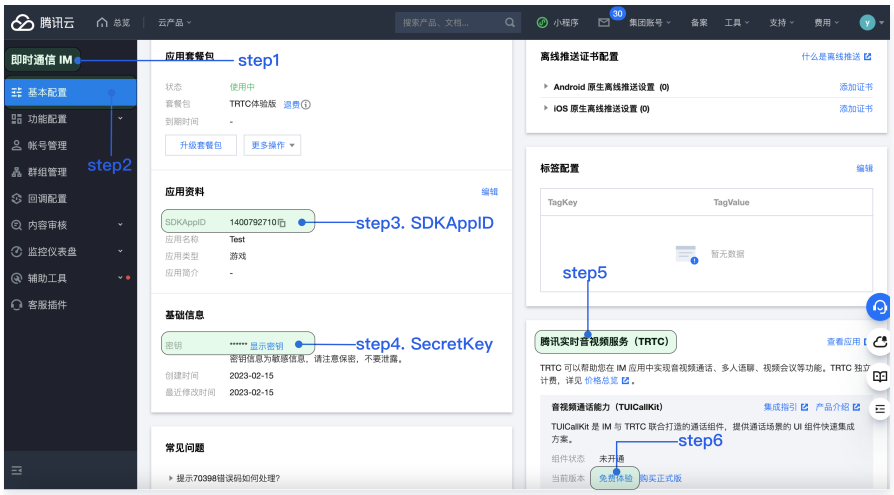
Windows 端

```
npm i @tencentcloud/call-uikit-wechat@1.4.4
```

```
xcopy node_modules\@tencentcloud\call-uikit-wechat .\wxcomponents\TUICallKit /i /e
```

步骤三：获取 SDKAppID & SecretKey

1. 登录到 即时通信 IM 控制台，单击**创建新应用**，在弹出的对话框中输入您的**应用名称**，并单击**确定**。
2. 单击创建的应用，进入**基本配置**页面，**获取 SDKAppID、SecretKey**。
3. 在基本配置页面，单击**免费体验**即可开通 TUICallKit 的 7 天免费试用服务。
4. 如果需要正式应用上线，可以单击 购买正式版 即可进入购买页面。

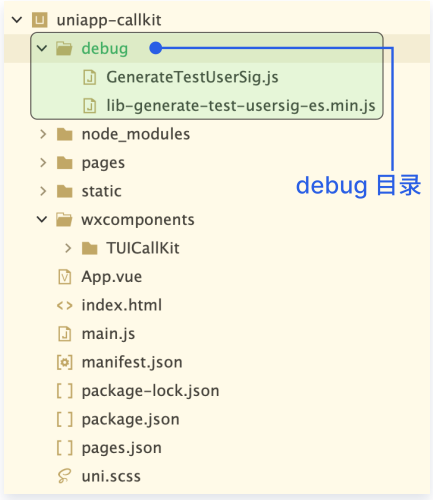


步骤四：获取 UserSig，修改 App.vue 文件

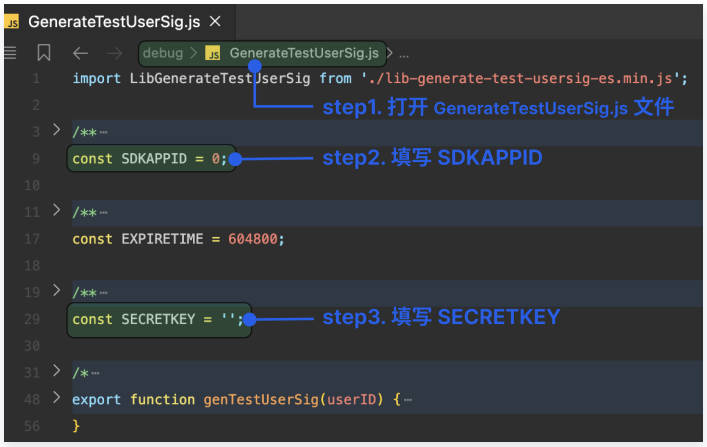
客户端生成

由于 UserSig 有时效性，如果您需要长期使用 TUICallKit，推荐您采用该方法。

1. 单击下载 debug 文件夹，将 debug 目录复制到您的项目，如下图所示：



2. 修改 `GenerateTestUserSig.js` 文件 的 SDKAPPID 以及 SECRETKEY。



3. 修改 `App.vue` 文件。

Vue3 环境

```
<script setup>
    import { genTestUserSig } from "./debug/GenerateTestUserSig";
    const globalData = {
      SDKAppID: 0,
      userID: "",
      userSig: "",
    };
    uni.getUserSig = (userID) => {
      getApp().globalData.userID = userID;
      const { userSig } = genTestUserSig(getApp().globalData.userID);
      getApp().globalData.userSig = userSig;
      getApp().globalData.SDKAppID = genTestUserSig("").sdkAppID;
    };
</script>
```

Vue2 环境

```
<script>
    import { genTestUserSig } from './debug/GenerateTestUserSig';
    const Signature = genTestUserSig('');
    export default {
        globalData: {
            SDKAppID: Signature.sdkAppID,
            userID: '',
            userSig: ''
        },
        methods:{
            getUserSig(){
                const { userSig } = genTestUserSig(getApp().globalData.userID);
                getApp().globalData.userSig = userSig;
            }
        }
    };
</script>
```

控制台生成

1. 如果您想要快速体验 callkit，您可以通过控制台中的 **辅助工具** 生成一个临时可用的 UserSig。

2. 修改 App.vue 文件。

Vue3 环境

```
<script setup>
const globalData = {
  SDKAppID: 0,
  userID: '',
  userSig: '',
};
uni.getUserSig = (userID) => {
  getApp().globalData.userID = userID;
  getApp().globalData.SDKAppID = 0 ;   //填写上您的 SDKAppID
  getApp().globalData.userSig = ''     //填写上您从控制台获取到的 userSig
};
</script>
```

Vue2 环境

```
<script>
export default {
    globalData: {
        SDKAppID: '',
        userID: '',
        userSig: ''
    },
    methods:{
        getUserSig(){
            console.log("getUserSig")
        }
    }
};
```

```
</script>
```

> **注意：**
> 每个 userID 对应的 userSig 是唯一的，若使用控制台生成临时的 userSig ，请在登录第二个用户时记得替换 app.js 中的 userSig。

步骤五：调用 TUICallKit 组件

1. 新建通话页面 call。

2. 修改 `page.json` 文件。

```
{
    "pages": [ //pages数组中第一项表示应用启动页，参考：https://uniapp.dcloud.io/collocation/pages
        {
            "path": "pages/index/index",
            "style": {
                "navigationBarTitleText": "uni-app"
            }
        }
        ,{
            "path" : "pages/call/call",
            "style": {
                "navigationBarTitleText": "uni-app",
                "usingComponents": {  // 修改点
                    "tuicallkit": "/wxcomponents/TUICallKit/TUICallKit/TUICallKit"
                }
            }
        }
    ],
    "globalStyle": {
        "navigationBarTextStyle": "black",
        "navigationBarTitleText": "uni-app",
        "navigationBarBackgroundColor": "#F8F8F8",
```

```
            "backgroundColor": "#F8F8F8"
        },
        "uniIdRouter": {}
    }
```

3. 修改 `index.vue` 文件。

Vue3 环境

```
<template>
  <view class="container">
    <view class="counter-warp">
      <view class="box">
        <view class="list-item">
          <label class="list-item-label">用户ID</label>
          <input
            class="input-box"
            type="text"
            v-model="userID"
            placeholder="请输入用户ID"
            placeholder-style="color:#BBBBBB;"
          />
        </view>
        <view class="login">
            <button class="loginBtn" @click="loginHandler">登录</button>
        </view>
      </view>
    </view>
  </view>
</template>
<script setup>
let userID = "";
const loginHandler = () => {
  uni.getUserSig(userID);
  uni.navigateTo({
    url: "../call/call",
  });
};
</script>
<style>
.box {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
input {
  display: flex;
  font-size: 20px;
}
.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
```

```
}
.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

Vue2 环境

```
<template>
  <view class="container">
    <view class="counter-warp">
      <view class="box">
        <view class="list-item">
          <label class="list-item-label">用户ID</label>
          <input
            class="input-box"
            type="text"
            v-model="userID"
            placeholder="请输入用户ID"
            placeholder-style="color:#BBBBBB;"
          />
        </view>
        <view class="login">
           <button class="loginBtn" @click="loginHandler">登录</button>
        </view>
      </view>
    </view>
  </view>
</template>
<script>
export default {
  data() {
    return {
      userID: "",
    };
  },
  methods: {
    loginHandler() {
      getApp().globalData.userID = this.userID;
      getApp().getUserSig();
      uni.navigateTo({
        url: "../call/call",
      });
    },
  },
};
</script>
<style>
.box {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
```

```
  align-items: center;
  justify-content: center;
}
input {
  display: flex;
  font-size: 20px;
}
.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
}
.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

4. 修改 `call.vue` 文件。

Vue3 环境

```
<template>
  <view class="container">
    <tuicallkit ref="TUICallKit"></tuicallkit>
    <view class="trtc-calling-index-search">
      <view class="search">
        <view class="input-box">
          <input
            class="input-search-user"
            :value="userIDToSearch"
            maxlength="11"
            type="text"
            v-on:input="userIDToSearchInput"
            placeholder="搜索用户ID"
          />
        </view>
        <view class="btn-search" @click="call">呼叫</view>
      </view>
      <view class="search-selfInfo">
        <label class="search-selfInfo-label">您的ID</label>
        <view class="search-selfInfo-phone">
          {{ data.userID }}
        </view>
      </view>
    </view>
  </view>
</template>

<script setup>
import { nextTick, shallowRef, reactive } from "vue";
import { onUnload, onLoad } from "@dcloudio/uni-app";
let TUICallKit = shallowRef(null);
```

```
const data = reactive({
  userID: "",
  userIDToSearch: "",
});

const userIDToSearchInput = (e) => {
  data.userIDToSearch = e.detail.value;
};

const call = async () => {
  try {
    await TUICallKit.value.call({
      userID: data.userIDToSearch,
      type: 2,
    });
  } catch (error) {
    console.error(error);
  }
};

onLoad(() => {
  data.userID = getApp().globalData.userID;
  nextTick(() => {
    TUICallKit.value.init({
      sdkAppID: getApp().globalData.SDKAppID,
      userID: getApp().globalData.userID,
      userSig: getApp().globalData.userSig,
    });
  });
});
onUnload(() => {
  TUICallKit.value.destroyed();
});
</script>

<style>
.trtc-calling-index {
  width: 100vw;
  height: 100vh;
  color: white;
  display: flex;
  flex-direction: column;
}

.trtc-calling-index-search > .search {
  width: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-sizing: border-box;
  padding: 16px;
}

.btn-search {
  text-align: center;
  width: 60px;
```

```
  height: 40px;
  line-height: 40px;
  background: #0a6cff;
  border-radius: 20px;
  color: aliceblue;
}

.search-selfInfo {
  position: relative;
  padding: 0 28px;
  font-size: 14px;
  color: #333333;
  font-weight: 400;
  display: flex;
  align-items: center;
}

.search-selfInfo-phone {
  padding-left: 8px;
}
</style>
```

Vue2 环境

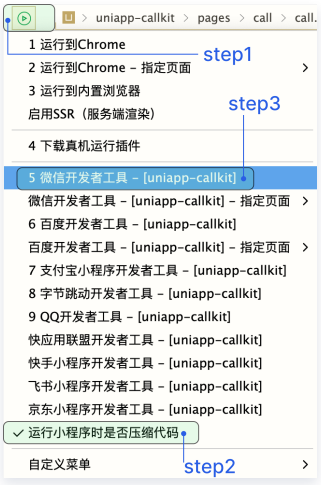
```
<template>
  <view class="container">
    <tuicallkit ref="TUICallKit"></tuicallkit>
    <view class="trtc-calling-index-search">
      <view class="search">
        <view class="input-box">
          <input
            class="input-search-user"
            :value="userIDToSearch"
            maxlength="11"
            type="text"
            v-on:input="userIDToSearchInput"
            placeholder="搜索用户ID"
          />
        </view>
        <view class="btn-search" @click="call">呼叫</view>
      </view>
      <view class="search-selfInfo">
        <label class="search-selfInfo-label">您的ID</label>
        <view class="search-selfInfo-phone">
          {{ userID }}
        </view>
      </view>
    </view>
  </view>
</template>

<script>
export default {
  data() {
    return {
```

```
      userIDToSearch: "",
      userID: "",
    };
  },
  methods: {
    userIDToSearchInput(e) {
      this.userIDToSearch = e.detail.value;
    },
    async call() {
      try {
        await this.$refs.TUICallKit.call({
          userID: this.userIDToSearch,
          type: 2,
        });
      } catch (error) {
        console.error(error)
      }
    },
  },
  onLoad() {
    this.userID = getApp().globalData.userID;
    this.$nextTick(() => {
      this.$refs.TUICallKit.init({
        sdkAppID: getApp().globalData.SDKAppID,
        userID: getApp().globalData.userID,
        userSig: getApp().globalData.userSig,
      });
    });
  },
  onUnload() {
    this.$refs.TUICallKit.destroyed();
  },
};
</script>

<style>
.trtc-calling-index {
  width: 100vw;
  height: 100vh;
  color: white;
  display: flex;
  flex-direction: column;
}

.trtc-calling-index-search > .search {
  width: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-sizing: border-box;
  padding: 16px;
}

.btn-search {
  text-align: center;
  width: 60px;
```

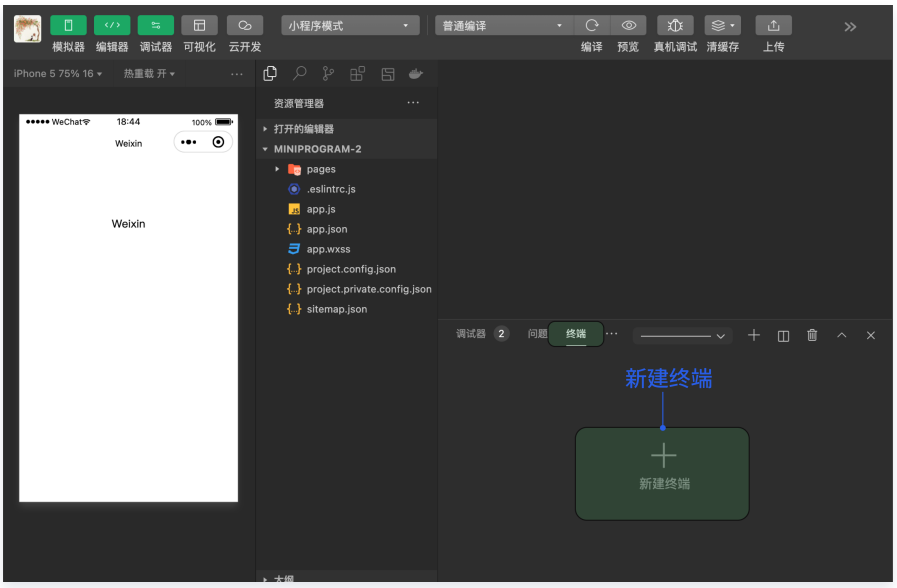
```
  height: 40px;
  line-height: 40px;
  background: #0a6cff;
  border-radius: 20px;
  color: aliceblue;
}

.search-selfInfo {
  position: relative;
  padding: 0 28px;
  font-family: PingFangSC-Regular;
  font-size: 14px;
  color: #333333;
  font-weight: 400;
  display: flex;
  align-items: center;
}

.search-selfInfo-phone {
  padding-left: 8px;
}
</style>
```

步骤六：运行到微信开发者工具

1. 运行到微信开发者工具。



2. 新建终端。



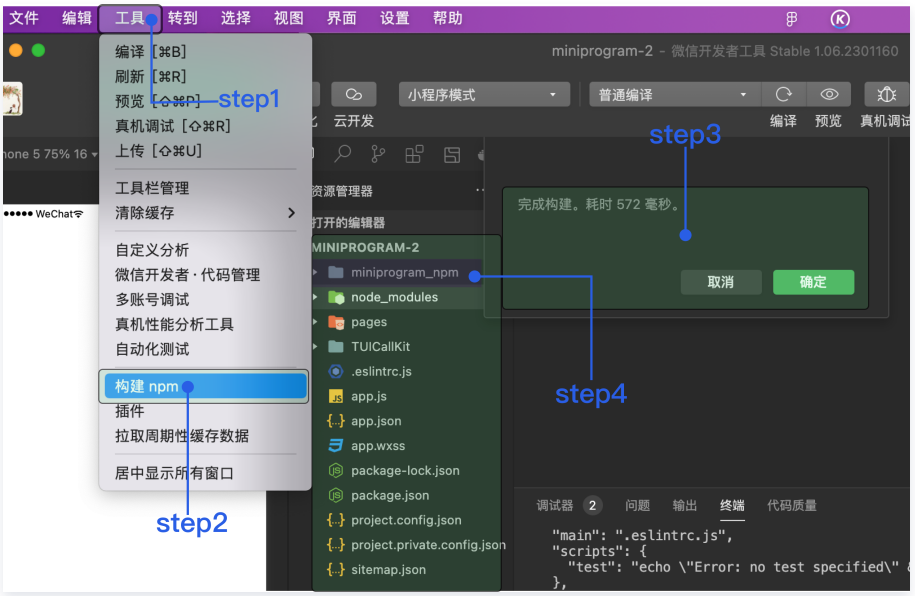
3. 初始化 package.json 文件，安装相关依赖包。

> **警告：**
> 因为 HBuilder 运行到小程序后，项目中的 package.json、node_modules 都不存在了。需要**微信开发者工具**的终端里，重新安装依赖。
>
> ```
> ▸ 页面 [wxcomponents/TUICallKit/TUICallKit/TUICallKit]错误:
> Error: module 'wxcomponents/TUICallKit/TUICallKit/tuicall-engine-wx.js' is not defined, require args is 'tuicall-engine-wx'
> ```

```
npm init -y
```

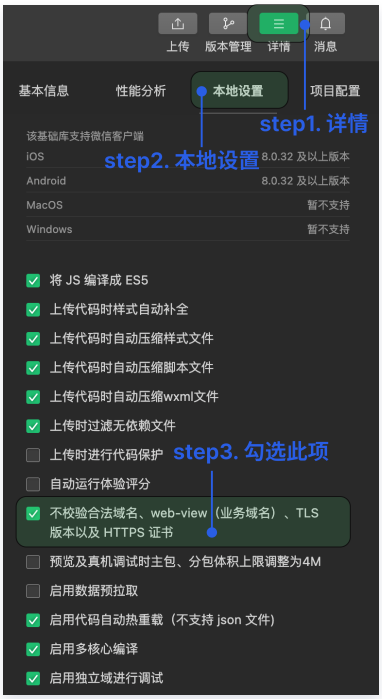
```
npm i @tencentcloud/call-uikit-wechat@1.4.4
```

4. 微信开发者工具**构建 npm**，新增 miniprogram_npm 目录。目录如下：



步骤七：编译运行

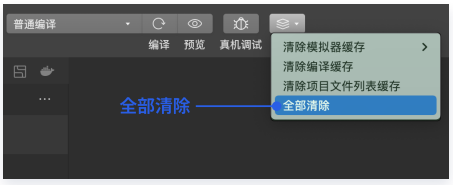
1. 请在本地设置里面勾选上“不校验合法域名、web-view (业务域名)、 TLS 版本以及 HTTPS 证书”。



警告：

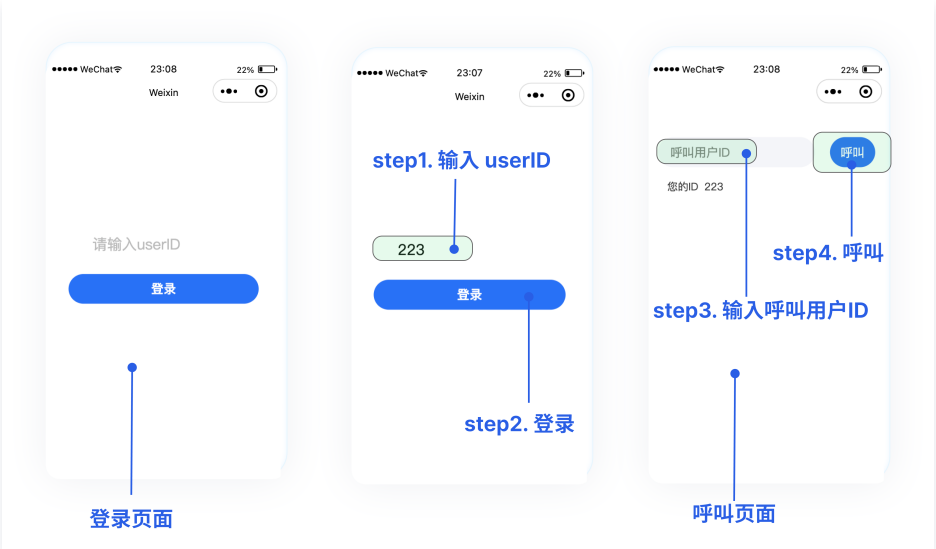
如果不勾选该条目，则会在控制台出现如下错误。

2. 单击**清缓存 > 全部清除**，避免开发者工具的缓存造成渲染异常。



3. 编译小程序。

4. 该项目快速集成后的预期效果图。



步骤八：拨打您的第一通电话

1. 请单击预览，扫描二维码，在真机环境使用小程序。

2. 登录后，请输入呼叫用户 ID，拨打您的第一通电话。具体效果如下图所示：

> ⚠ **注意：**
> 第一次使用小程序通话，需要获取摄像头和麦克风权限。

更多特性

设置昵称和头像

如果您需要自定义昵称和头像，可以使用如下接口进行更新：

Vue3 环境

```
TUICallKit.value.setSelfInfo("昵称", "头像 URL");
```

> ⚠ **注意：**
> - 获取组件必须使用 shallowRef，不然会有 this 指向报错的问题，后续 uni-app 官方会升级修复这个问题，就可以正常使用 ref 获取。
> - 在调用组件方法的时候需要加上 value。

Vue2 环境

```
this.$refs.TUICallKit.setSelfInfo("昵称", "头像 URL");
```

自定义铃声

如果您需要自定义来电铃音，可以通过如下接口进行设置：

- 传入本地铃声文件应为绝对路径。
- 如需恢复默认铃声，`filePath` 传空即可。

Vue3 环境

```
TUICallKit.value.setCallingBell("filePath");
```

Vue2 环境

```
this.$refs.TUICallKit.setCallingBell("filePath");
```

群组通话

如果您需要实现群组（即多人）音视频通话，可以通过如下接口进行设置：

Vue3 环境

```
TUICallKit.value.groupCall({
```

```
    userIDList: ["jane", "mike", "tommy"],
    type: 1,    // 1-语音通话 2-视频通话
    groupID: "12345678"
});
```

Vue2 环境

```
this.$refs.TUICallKit.groupCall({
    userIDList: ["jane", "mike", "tommy"],
    type: 1,    // 1-语音通话 2-视频通话
    groupID: "12345678"
});
```

API 文档

小程序 TUICallKIt API 文档请参考：TUICallKit API 。

常见问题

什么是 SDKAppID、SecretKey?

- **SDKAppID**：IM 的应用 ID，用于业务隔离，即不同的 SDKAppID 的通话彼此不能互通。
- **Secretkey**：IM 的应用密钥，需要和 SDKAppID 配对使用，用于签出合法使用 IM 服务的鉴权用票据 UserSig。

什么是 UserSig，如何生成 UserSig?

- UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。
- UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig 。

调用 call 方法提示报错 undefined，如何处理?

1. 检查 page.json 文件中 TUICallKit 的路径是否正确。

```
  {
      "path" : "pages/call/call",
       "style": {
            "navigationBarTitleText": "uni-app",
             "usingComponents": {   // 修改点
                  "tuicallkit": "/wxcomponents/TUICallKit/TUICallKit/TUICallKit"
             }
       }
}
```

2. 不要在 TUICallKit 的 dom 标签上使用 v-if 来进行条件渲染。

```
  <TUICallKit
    id="TUICallKit-component"
  ></TUICallKit>
```

> **警告：**
> 使用 v-if 条件渲染指令隐藏节点后，该节点上的方法和事件也会被隐藏，无法正常触发。因此，无法通过隐藏节点上的方法来操作节点或获取数据。如果需要在隐藏节点中执行一些操作，可以考虑使用 v-show 来代替 v-if，或者通过其他方式实现隐藏和显示的效果。

怎样监听通话过程中的事件?

1. 导入 tuicall-engine-wx 的事件模块。

```
import { EVENT } from 'tuicall-engine-wx';
```

2. 绑定监听事件(以通话结束事件为例)。

```
// 通话结束
wx.$TUICallEngine.on(EVENT.CALL_END, this.handleCallingEnd, this);
```

3. 通话结束会触发回调函数 handleCallingEnd。

```
// 通话结束
handleCallingEnd(event) {
  console.log(event)
},
```

4. 移除监听。

```
// 通话结束
wx.$TUICallEngine.off(EVENT.CALL_END, this.handleCallingEnd);
```

更多的事件监听请参考 TUICallEvent 。

> **说明:**
> 如需更多帮助，可以加入我们的 TUICallKit 技术交流 QQ 群：**605115878**，进行技术交流和产品沟通。