TRTC 云助手

模块化方案

腾讯云

【版权声明】

©2013-2024 腾讯云版权所有

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

【商标声明】

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

【服务声明】

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

【联系我们】

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

文档目录

模块化方案
- 模块化方案概述
- 视频渲染控件样式布局
- 移动端应用保活方案
- 跨房 PK 连麦方案
- 系统来电监听与处理
- 客户端合流方案

版权所有：腾讯云计算（北京）有限责任公司

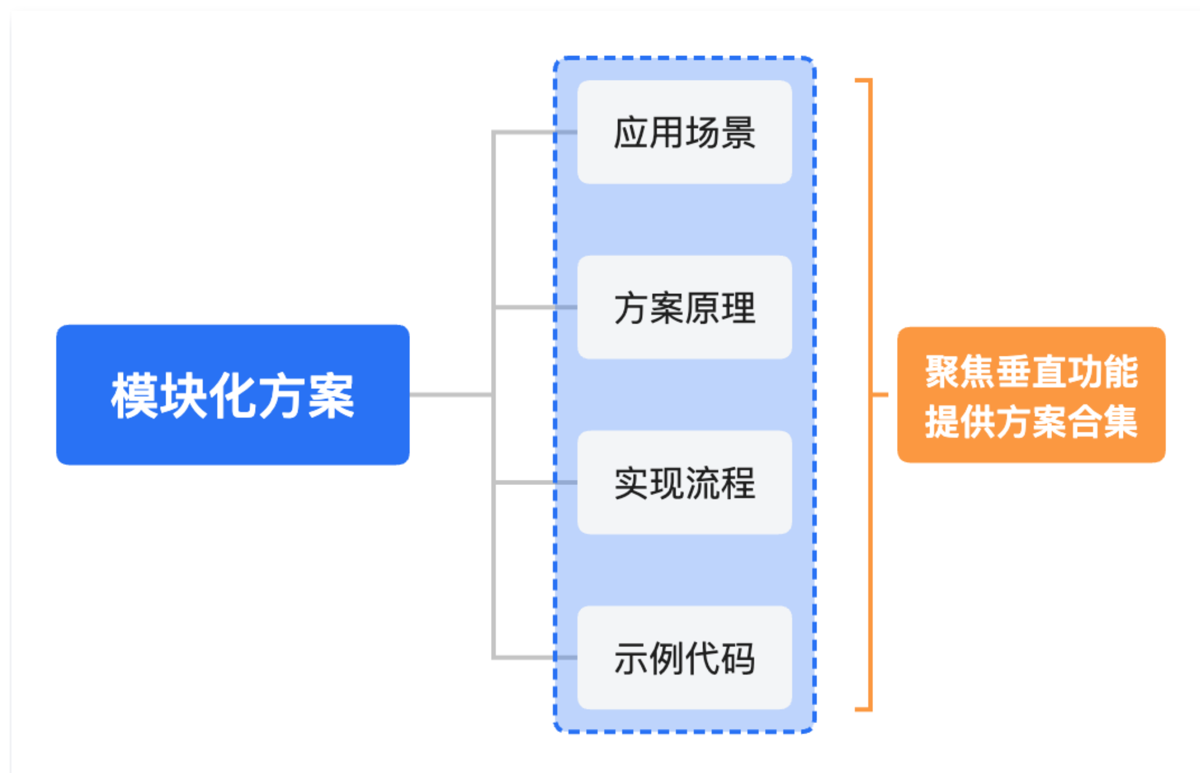
模块化方案

模块化方案概述

最近更新时间：2024-04-18 18:27:51

概述

模块化方案独立于场景化方案之外，更加聚焦音视频通信领域的垂直功能，提供对应的解决方案合集。目前涵盖了 跨房 PK 连麦方案 、客户端合流方案 等，可以帮助您深入理解单个功能模块的应用场景、方案原理和实现流程等，助力业务快速进行方案的选型和落地。



视频渲染控件样式布局

最近更新时间：2024-07-24 16:21:42

音视频场景会高频使用到视频渲染控件，例如直播播放、点播播放、本地摄像头预览等。为了更加美观，我们可以为视频渲染控件设置个性化的样式布局，例如阴影和圆角效果。下面我们将分别介绍 Android 平台 以及 iOS 平台 的视频渲染控件样式布局实现方式。

Android 平台

腾讯云实时音视频、直播点播播放器等产品均会涉及到视频渲染控件的使用。在 Android 平台，通常使用腾讯云自定义的视频渲染控件 TXCloudVideoView。该控件本身不支持直接设置阴影、圆角等效果，因此我们通常会借助 CardView 视图组件来实现个性化的样式布局，实现步骤如下。

步骤一：添加依赖

在 build.gradle 文件中添加 CardView 的依赖。

```
dependencies {
    implementation 'androidx.cardview:cardview:1.0.0'
}
```

步骤二：自定义 CardView

在 XML 布局文件中，可以像使用其他视图组件一样使用 CardView。您可以通过 card_view 命名空间中的属性来自定义 CardView 的外观：

- cardCornerRadius：设置卡片的圆角半径。
- cardElevation：设置卡片的阴影高度。
- cardBackgroundColor：设置卡片的背景颜色。

```
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    card_view:cardCornerRadius="12dp"
    card_view:cardElevation="6dp"
    card_view:cardBackgroundColor="@color/white">

    <!-- 内部布局 -->
```

```
</androidx.cardview.widget.CardView>
```

步骤三：添加 TXCloudVideoView

使用 CardView 包裹 TXCloudVideoView，从而使得腾讯云视频渲染控件实现阴影、圆角等效果。

```
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    card_view:cardCornerRadius="12dp"
    card_view:cardElevation="6dp"
    card_view:cardBackgroundColor="@color/white">

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

</androidx.cardview.widget.CardView>
```

在代码中动态创建和操作 CardView

当然，除了在布局文件中使用 CardView，也可以在代码中动态创建和操作 CardView，示例代码如下。

```
// 初始化 CardView
CardView cardView = new CardView(context);

// 设置 CardView 的属性
cardView.setRadius(12f);  // 设置圆角半径
cardView.setCardElevation(6f);  // 设置阴影高度
cardView.setCardBackgroundColor(ContextCompat.getColor(this,
R.color.white));  // 设置背景颜色

// 初始化 TXCloudVideoView
TXCloudVideoView txCloudVideoView = new TXCloudVideoView(context);

// 将 TXCloudVideoView 添加到 CardView
cardView.addView(txCloudVideoView);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 将 CardView 添加到父布局
parentLayout.addView(cardView);
```

iOS 平台

腾讯云实时音视频、直播和点播播放器等产品均会涉及到视频渲染控件的使用。在 iOS 平台，可以直接使用 UIView 来设置圆角和阴影等效果。

设置圆角效果

如果您需要单独设置圆角效果，可参考如下代码示例实现。

```
UIView *contentView = [[UIView alloc] initWithFrame:CGRectMake(50, 100,
200, 200)];
contentView.backgroundColor = [UIColor whiteColor];
contentView.layer.cornerRadius = 20;
// 剪切以显示圆角
contentView.clipsToBounds = YES;
// 添加到主视图
[self.view addSubview:contentView];
// 开启本地摄像头的预览画面
[self.trtcCloud startLocalPreview:self.isFrontCamera view:contentView];
```

设置圆角和阴影

如果您需要同时设置圆角和阴影效果，此时需要使用两个视图：一个视图用于显示阴影，另一个视图用于显示内容并设置圆角。这样可以确保圆角和阴影都能正确显示。下面是详细的步骤和代码示例。

```
// 容器视图，用于阴影
UIView *shadowView = [[UIView alloc] initWithFrame:CGRectMake(50, 100,
200, 200)];
shadowView.layer.shadowColor = [UIColor redColor].CGColor;
shadowView.layer.shadowOpacity = 0.5;
shadowView.layer.shadowOffset = CGSizeMake(5, 5);
shadowView.layer.shadowRadius = 10;
shadowView.backgroundColor = [UIColor clearColor];
shadowView.clipsToBounds = NO; // 保持为NO以显示阴影

// 内容视图，用于显示内容和设置圆角
UIView *contentView = [[UIView alloc] initWithFrame:shadowView.bounds];
contentView.backgroundColor = [UIColor whiteColor];
contentView.layer.cornerRadius = 20;
contentView.clipsToBounds = YES; // 剪切以显示圆角
```

版权所有：腾讯云计算（北京）有限责任公司

```
[shadowView addSubview:contentView]; // 将内容视图添加到阴影视图
[self.view addSubview:shadowView]; // 将阴影视图添加到主视图

// 开启本地摄像头的预览画面
[self.trtcCloud startLocalPreview:self.isFrontCamera view:contentView];
```

版权所有：腾讯云计算（北京）有限责任公司

移动端应用保活方案

最近更新时间：2024-07-23 10:42:42

对于涉及音视频采集和播放的移动端应用，通常需要进行额外的保活处理，否则应用在后台运行时会受到一定的功能性限制，甚至在后台运行一段时间后被系统强制终止运行。下面我们分别介绍 Android 应用保活方案，以及 iOS 应用保活方案 。

Android 应用保活方案

目前 Android 端常用的保活方式是启动前台服务。前台服务是一种特殊的服务，它在运行时会显示一个持续的通知，以告知用户该服务正在运行。由于前台服务的通知是持续可见的，系统会认为这是一个高优先级的任务，应用可以在后台持续运行，而不容易被系统终止。下面介绍前台服务的实现方式及步骤。

步骤一：声明权限

在 AndroidManifest.xml 文件中添加以下权限声明。

```
<!-- 允许应用使用前台服务 -->
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />

<!-- 如果应用需要在前台服务中使用摄像头 -->
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE_CAMERA" />

<!-- 如果应用需要在前台服务中使用麦克风 -->
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE_MICROPHONE" />
```

> **注意：**
> 若您的项目设置 targetSdkVersion 34 及以上，且需要在前台服务中使用摄像头和麦克风，则 `FOREGROUND_SERVICE_CAMERA` 和 `FOREGROUND_SERVICE_MICROPHONE` 权限声明是必须的。

步骤二：创建服务类

创建一个继承自 Service 的类，并在其中实现前台服务的逻辑。

```
public class MyForegroundService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
```

```
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // 创建通知
        Notification notification = createNotification();

        // 启动前台服务
        startForeground(1, notification);

        // 处理其他逻辑
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // 停止前台服务
        stopForeground(true);
    }

    private Notification createNotification() {
        // 创建通知渠道（仅适用于Android 8.0及以上版本）
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    "CHANNEL_ID",
                    "前台服务通知",
                    NotificationManager.IMPORTANCE_DEFAULT
            );
            NotificationManager manager =
getSystemService(NotificationManager.class);
            if (manager != null) {
                manager.createNotificationChannel(channel);
            }
        }

        // 创建通知
        return new NotificationCompat.Builder(this, "CHANNEL_ID")
                .setContentTitle("前台服务")
                .setContentText("服务正在运行")
                .setSmallIcon(R.drawable.ic_notification)
                .build();
    }
}
```

步骤三：声明服务

在 AndroidManifest.xml 文件中声明服务。

```
<service
    android:name=".MyForegroundService"
    android:enabled="true"
    android:exported="false"

android:foregroundServiceType="mediaPlayback|mediaProjection|microphone|
camera" />
```

> **注意：**
> 您可以通过 `android:foregroundServiceType` 属性指定前台服务需要使用的服务类型，以确保退后台可以保持正常的服务功能。
> - `mediaPlayback` 服务用于媒体播放。
> - `mediaProjection` 服务用于媒体投影。
> - `microphone` 服务用于使用麦克风。
> - `camera` 服务用于使用摄像头。

步骤四：启动前台服务

在需要启动前台服务的地方，按需启动前台服务。

```
NotificationManagerCompat notificationManager =
NotificationManagerCompat.from(this);
boolean areNotificationsEnabled =
notificationManager.areNotificationsEnabled();
if (!areNotificationsEnabled) {
    // 提示用户启用通知权限
    Toast.makeText(this, "请启用通知权限以确保服务正常运行",
Toast.LENGTH_LONG).show();
    // 引导用户到设置页面
    Intent intent = new
Intent(Settings.ACTION_APP_NOTIFICATION_SETTINGS)
            .putExtra(Settings.EXTRA_APP_PACKAGE, getPackageName());
    startActivity(intent);
} else {
    // 启动前台服务
    Intent serviceIntent = new Intent(this, MyForegroundService.class);
```

版权所有：腾讯云计算（北京）有限责任公司

```
    ContextCompat.startForegroundService(this, serviceIntent);
}
```

> ⚠ **注意：**
> 为了确保前台服务能够正常运行，建议在启动前台服务之前，检查通知权限是否被禁用。如果被禁用，可以提示用户启用通知权限。

步骤五：停止前台服务

可从外部组件（例如 Activity 或 BroadcastReceiver）中停止前台服务。

```
// 创建服务的Intent
Intent serviceIntent = new Intent(this, MyForegroundService.class);

// 停止服务
stopService(serviceIntent);
```

在大部分移动设备上，针对启动前台服务的应用，用户在最近应用列表里划卡强杀，前台服务会同时终止，应用会完全停止运行。但在部分境外品牌移动设备上（例如 Google Pixel 系列和 SAMSUNG A 系列），划卡强杀后应用并不会完全停止运行，前台服务仍然处于活跃状态，这会导致用户还能够听到媒体播放。
针对此类设备，可以通过实现以下两种方案，从而避免应用被强杀后仍然播放媒体声音的现象。

方案一：在服务声明中定义 stopWithTask 属性

添加 `android:stopWithTask="true"` 属性值，服务会在任务被移除时立即停止。

```
<service
    android:name=".MyForegroundService"
    android:enabled="true"
    android:exported="false"
    android:stopWithTask="true"
    android:foregroundServiceType="mediaPlayback|microphone" />
```

方案二：在 Service 层监听 onTaskRemoved 回调

监听 onTaskRemoved，当任务被移除时，该方法会被回调，您可以在这里执行清理操作或保存数据的逻辑。

```
@Override
public void onTaskRemoved(Intent rootIntent) {
    super.onTaskRemoved(rootIntent);
    // 例如在这里执行TRTC退房，避免继续进行音频采集和播放
```

版权所有：腾讯云计算（北京）有限责任公司

```
    TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(this);
    mTRTCCloud.exitRoom();
}
```

> **注意：**
> 以上两种方案任选其一即可，当设置 `android:stopWithTask="true"` 之后， `onTaskRemoved` 方法将不会被回调。

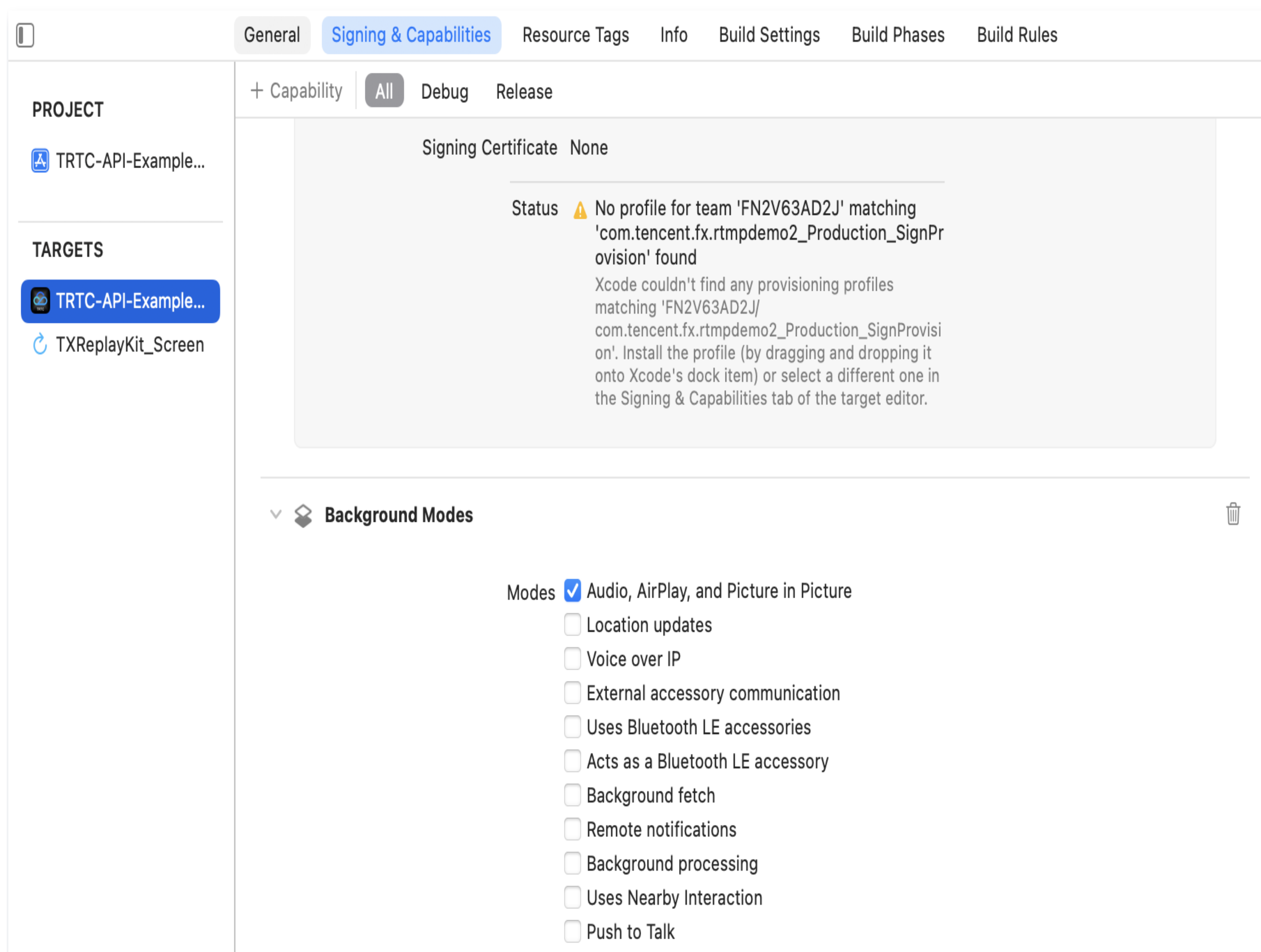
iOS 应用保活方案

苹果对应用在后台的行为有严格的限制，以保护用户的隐私和设备的电池寿命。通常可以通过启用特定的后台模式（Background Modes），从而在一定程度上实现应用保活，允许应用在后台播放音频或视频。

启用后台模式

Xcode 启用后台模式（Background Modes）的步骤如下：

1. 打开您的 Xcode 项目。
2. 选择您的项目文件（通常在项目导航器的顶部）。
3. 在项目文件中，选择您的目标（**Targets**）。
4. 在目标设置中，选择 **Signing&Capabilities** 选项卡。
5. 找到 **Background Modes** 选项，勾选 **Audio, AirPlay, and Picture in Picture** 模式。



常见问题

1. **在语音通话或直播场景中，主播将应用退至后台或锁屏，插拔耳机导致音频采集和播放无声**

   在 SDK 默认的音频策略下，系统音量类型处于自动切换模式，即“麦上通话，麦下媒体”。同时音量类型也会随音频路由的变化而变化，例如插入耳机音量类型会由通话音量切换至媒体音量。因此，在自动切换模式下，主播插拔耳机会导致系统音量类型的切换，此时系统需要重启音频驱动。而 iOS 系统在后台或锁屏状态下重启音频驱动有概率会失败，所以会导致音频采集和播放无声。

   针对这类问题，可以通过固定系统音量类型来规避，例如指定全程通话音量或全程媒体音量。

```
// 指定全程通话音量
[self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeVOIP];

// 指定全程媒体音量
[self.trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
```

2. **在视频通话或直播场景中，主播将应用退至后台或锁屏，远端观众拉流画面黑屏但声音正常**

   苹果系统严格禁止应用在后台采集视频。即使您启用了后台模式，应用在进入后台后，摄像头仍然会自动停止工作。这是为了保护用户的隐私，防止应用在未经用户同意的情况下录制视频。因此，这类场景下的视频采集问题暂时无法避免，只能实现音频的正常采集和播放。

3. **观众进房时房间内无人推流，将应用退至后台或锁屏，后续房间内有人推流也无法正常接收**

   iOS 应用在切换至后台之前，如果没有启动 AudioUnit 采集或播放，则很快会被挂起，且无法人为唤醒，直到切换至前台。要解决此类问题，只需要在应用切换至后台之前，保持 AudioUnit 一直运行即可（播放静音数据）。具体实现方法可参考下方示例代码。

```
// 进房之后启用自定义音轨
[self.trtcCloud enableMixExternalAudioFrame:NO playout:YES];

// 退房之前关闭自定义音轨
[self.trtcCloud enableMixExternalAudioFrame:NO playout:NO];
```

跨房 PK 连麦方案

最近更新时间：2024-08-07 21:49:21

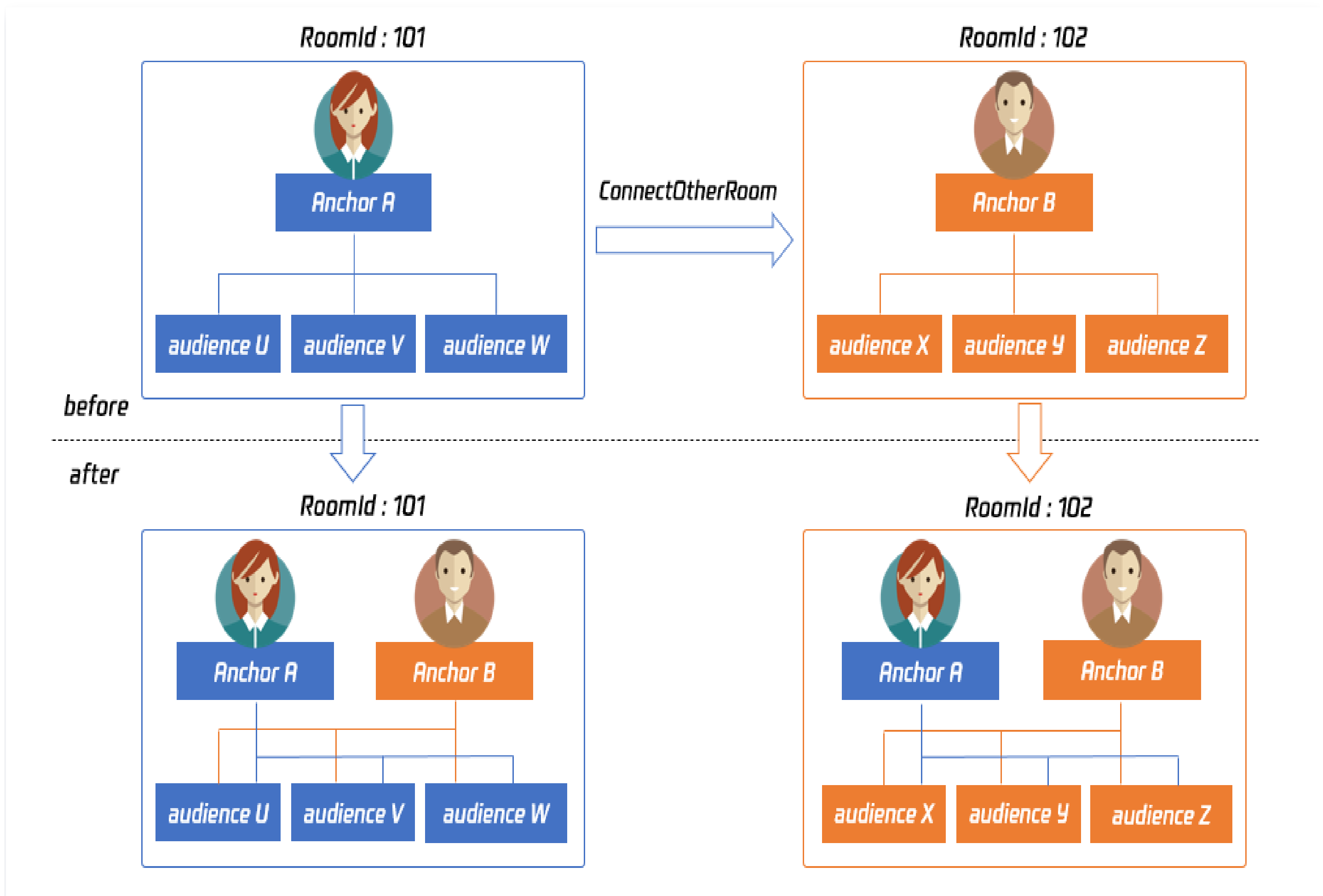
直播间里，为了增进直播气氛、快速吸粉，主播可以邀请其他直播间的主播进行连麦互动或在线 PK。连麦直播间内的观众可以同时收听或观看多个主播互动，能够增强互动直播的趣味性，激发观众刷榜送礼物的欲望。下面介绍三种不同跨房 PK 连麦方案的具体实现。

常规跨房 PK 连麦方案

适用场景：两个或多个房间 PK，房间主播数量较少的简单跨房连麦场景。

方案原理

默认情况下，只有同一个房间内的用户之间音视频可以互通，不同的房间之间的音视频流是相互隔离的。通过跨房连麦，可以将另一个房间中某个主播音视频流发布到自己所在的房间中，与此同时也会将自己的音视频流发布到目标主播的房间中。让身处两个不同房间中的主播进行跨房间的音视频流分享，从而让每个房间中的观众都能观看到这两个主播的音视频。



实现流程

当房间“101”中的主播 A 通过 ConnectOtherRoom 跟房间“102”中的主播 B 建立跨房通话后：

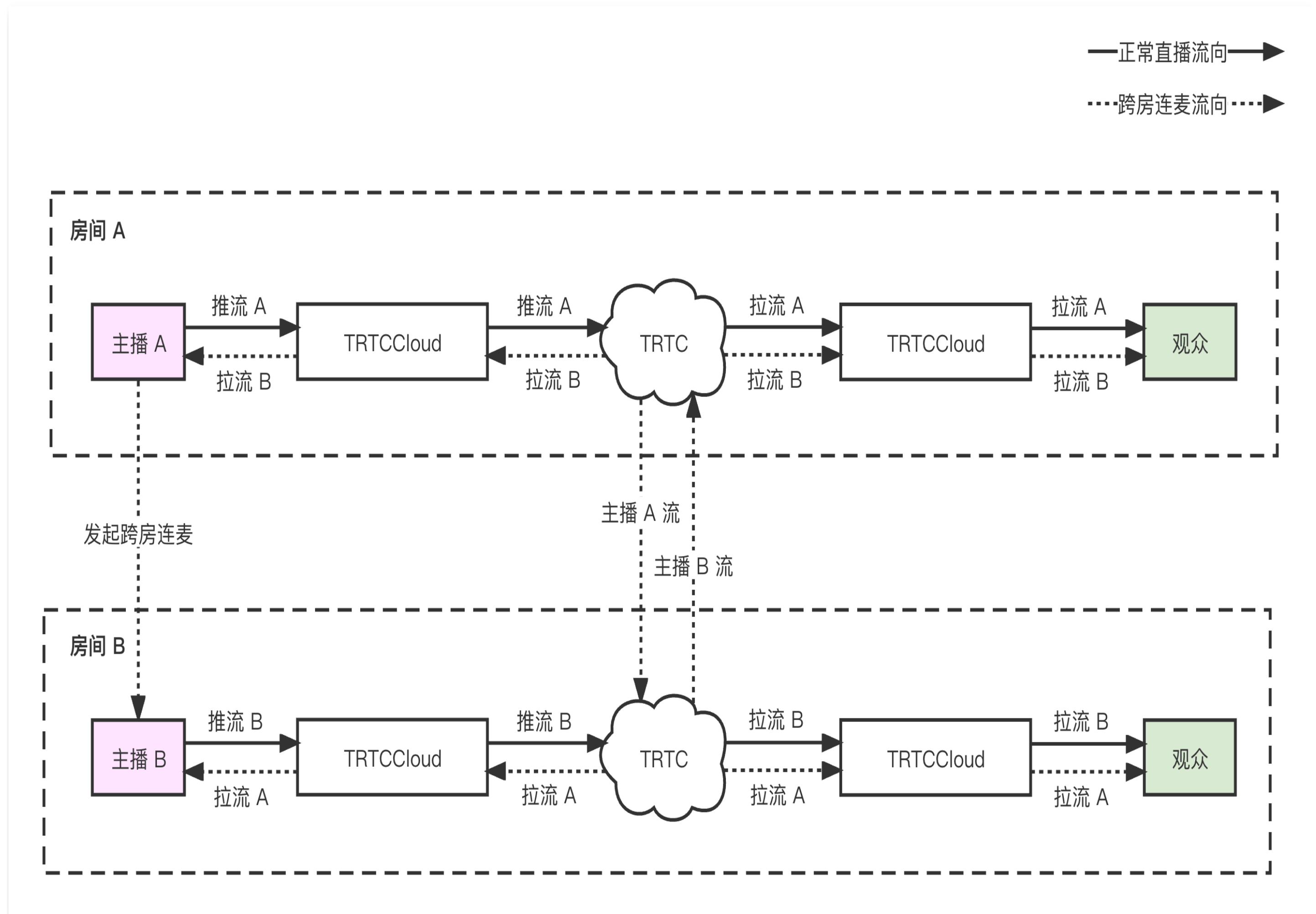
- 房间“101”中的用户都会收到主播 B 的 `onRemoteUserEnterRoom(B)` 和 `onUserVideoAvailable(B,true)` 这两个事件回调，即房间“101”中的用户都可以订阅主播 B 的音视频。
- 房间“102”中的用户都会收到主播 A 的 `onRemoteUserEnterRoom(A)` 和 `onUserVideoAvailable(A,true)` 这两个事件回调，即房间“102”中的用户都可以订阅主播 A 的音视频。

> **注意：**
> - 两个房间单主播跨房 PK，只需要其中一个房间的主播调用 `ConnectOtherRoom` 建立跨房连麦即可，请勿双向调用。
> - 主播可通过多次调用 `ConnectOtherRoom` 与多个房间的主播建立跨房连麦，目前限制单个主播最多和其他房间的 9 个主播进行跨房连麦。

版权所有：腾讯云计算（北京）有限责任公司

实时互动跨房连麦

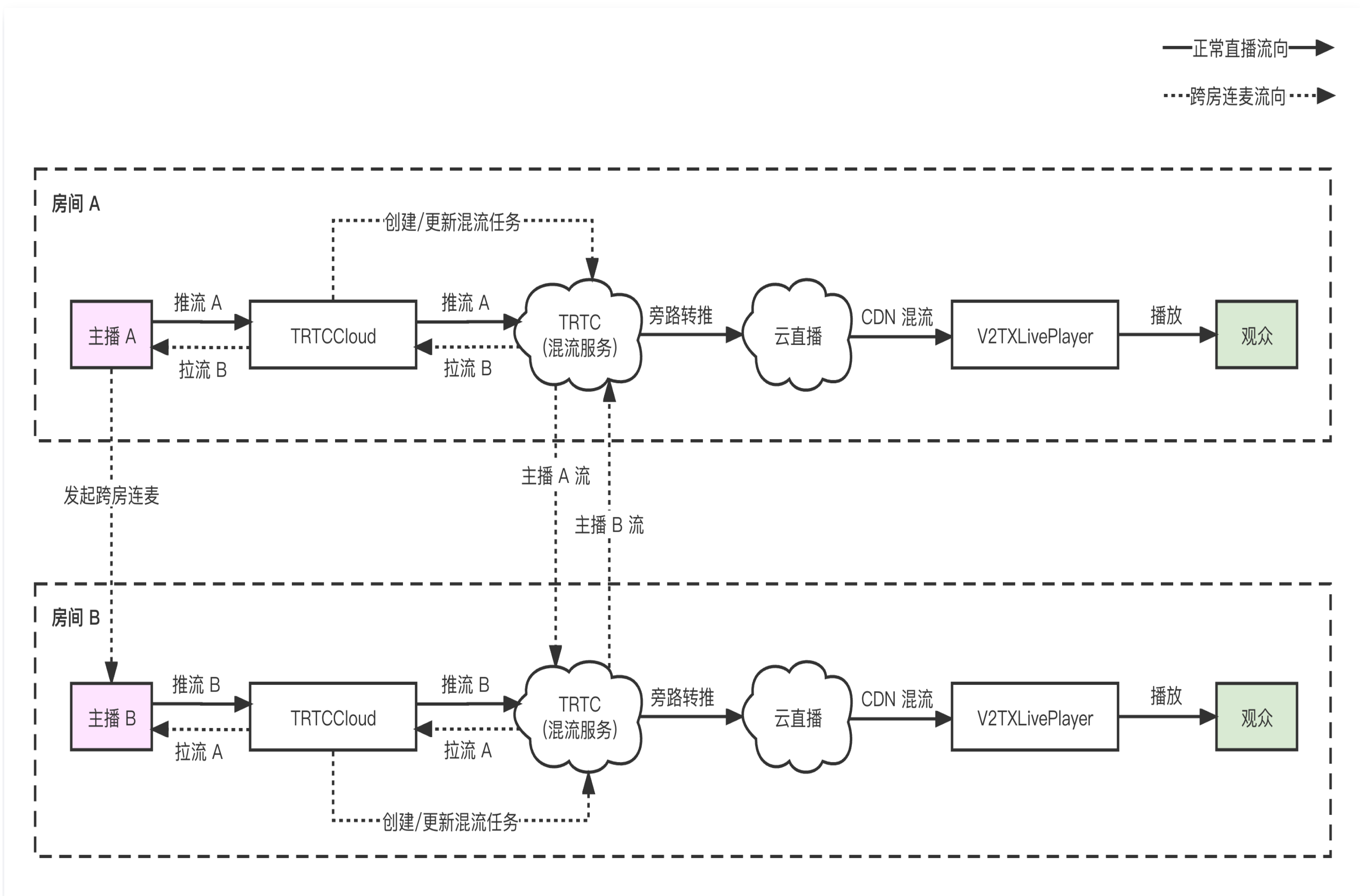
RTC 场景下的跨房连麦 PK 流程整体简单，主播和跨房连麦主播互相拉取 RTC 单流，观众同时拉取主播和跨房连麦主播的 RTC 单流，观众可独立控制主播和跨房连麦主播媒体流订阅逻辑。实时互动场景跨房连麦流程如下图所示。



> **注意：**
> 实时互动跨房连麦场景下，房间内观众可独立控制跨房连麦主播媒体流的订阅逻辑，也可由主播 更改跨房主播在本房间的上行能力 。

旁路直播跨房连麦

旁路直播场景下的跨房连麦 PK 流程相对复杂，主播和跨房连麦主播互相拉取 RTC 单流，CDN 观众拉取主播和跨房连麦主播的旁路混流，观众无法独立控制主播和跨房连麦主播媒体流订阅逻辑。旁路直播场景跨房连麦流程如下图所示。



> **注意：**
> 旁路直播跨房连麦场景下，CDN 观众无法独立控制跨房连麦主播媒体流的订阅逻辑，需由主播通过 更新发布媒体流 来统一控制。

示例代码

1. 任意一方发起跨房 PK 连麦。

Android

```
public void connectOtherRoom(String roomId, String userId) {
    try {
        JSONObject jsonObj = new JSONObject();
        // 以字符串房间号为例，数字房间号 key:roomId
        jsonObj.put("strRoomId", roomId);
        jsonObj.put("userId", userId);
        mTRTCCloud.ConnectOtherRoom(jsonObj.toString());
```

```
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

// 请求跨房连麦的结果回调
@Override
public void onConnectOtherRoom(String userId, int errCode, String
errMsg) {
    // userId:  要跨房连麦的另一个房间中的主播的用户 ID
    // errCode: 错误码，ERR_NULL 代表请求成功
    // errMsg:  错误信息
}
```

iOS

```
- (void)connectOtherRoom:(NSString *)roomId {
    NSMutableDictionary *jsonDict = [[NSMutableDictionary alloc] init];
    // 以字符串房间号为例，数字房间号 key:roomId
    [jsonDict setObject:roomId forKey:@"strRoomId"];
    [jsonDict setObject:self.userId forKey:@"userId"];
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict
options:NSJSONWritingPrettyPrinted error:nil];
    NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
    [self.trtcCloud connectOtherRoom:jsonString];
}

// 请求跨房连麦的结果回调
- (void)onConnectOtherRoom:(NSString *)userId errCode:
(TXLiteAVError)errCode errMsg:(NSString *)errMsg {
    // userId:  要跨房连麦的另一个房间中的主播的用户 ID
    // errCode: 错误码，ERR_NULL 代表请求成功
    // errMsg:  错误信息
}
```

注意：
- 跨房 PK 连麦的本地用户和对端用户必须都为主播角色，且必须都有音频或视频上行。
- 两个房间单主播跨房 PK，只需要其中一个房间的主播调用 `ConnectOtherRoom` 建立跨房连麦即可，请勿双向调用。

版权所有：腾讯云计算（北京）有限责任公司

2. 两个房间中的所有用户都会收到来自另一个房间中的 PK 主播的音视频流可用回调。

Android

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 某远端用户发布/取消了自己的音频
    // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 某远端用户发布/取消了主路视频画面
    if (available) {
        // 订阅远端用户的视频流，并绑定视频渲染控件
        mTRTCCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, view);
    } else {
        // 停止订阅远端用户的视频流，并释放渲染控件
        mTRTCCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

iOS

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了自己的音频
    // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了主路视频画面
    if (available) {
        // 订阅远端用户的视频流，并绑定视频渲染控件
        [self.trtcCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 停止订阅远端用户的视频流，并释放渲染控件
```

版权所有：腾讯云计算（北京）有限责任公司

```
        [self.trtcCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
    }
}
```

3. 任意一方退出跨房 PK 连麦。

Android

```
// 退出跨房连麦
mTRTCCloud.DisconnectOtherRoom();

// 退出跨房连麦的结果回调
@Override
public void onDisConnectOtherRoom(int errCode, String errMsg) {
    super.onDisConnectOtherRoom(errCode, errMsg);
}
```

iOS

```
// 退出跨房连麦
[self.trtcCloud disconnectOtherRoom];

// 退出跨房连麦的结果回调
- (void)onDisconnectOtherRoom:(TXLiteAVError)errCode errMsg:(NSString
*)errMsg {
}
```

注意：
跨房 PK 连麦的发起端和接收端任意一端均可调用 DisconnectOtherRoom 结束跨房 PK 连麦。

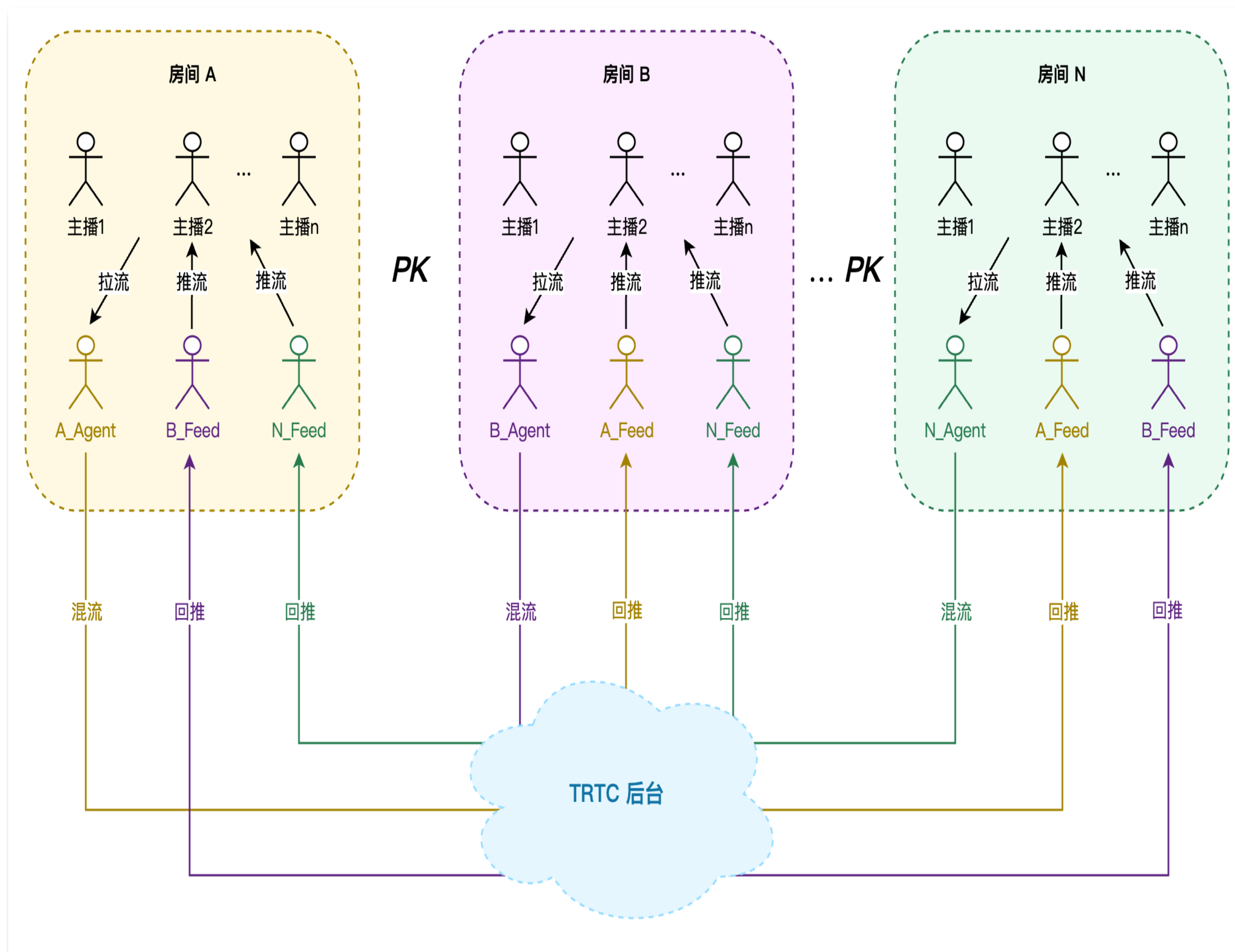
服务端跨房 PK 连麦方案

适用场景：多个房间 PK，每个房间有多个主播的纯服务端跨房连麦场景。

方案原理

服务端启动多个混流转推任务，每个转推任务都会拉起一个 Agent 机器人用户进入己方 TRTC 房间进行拉流，同时会拉起一个或多个 Feed 机器人用户将混合的音视频流回推到其他参与跨房 PK 连麦的 TRTC 房间。这样不同房间里的用户就可以通过订阅其他房间混流回推的音视频流，从而实现跨房 PK 连麦。

版权所有：腾讯云计算（北京）有限责任公司



实现流程

实时互动跨房连麦

1. 房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。
2. 房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
3. 业务后台同时启动 N 个混流回推房间任务 StartPublishCdnStream。
    - 任务一：A_Agent 机器人接收 A 房间主播媒体流，经 TRTC 后台混流后由 A_Feed 机器人回推到 B 房间和 N 房间。
    - 任务二：B_Agent 机器人接收 B 房间主播媒体流，经 TRTC 后台混流后由 B_Feed 机器人回推到 A 房间和 N 房间。
    - 任务 N：N_Agent 机器人接收 N 房间主播媒体流，经 TRTC 后台混流后由 N_Feed 机器人回推到 A 房间和 B 房间。

4. 房间 A、房间 B、房间 N 的用户互相拉取房间中混流回推的音视频流，开始进行跨房 PK。
5. 跨房 PK 结束，业务后台通过 TaskId 停止 N 个混流回推房间任务 StopPublishCdnStream 。

> 注意：
> - 本方案最多支持 11 个房间同时进行跨房 PK 连麦，每个房间最多支持 16 个主播同时参与连麦。
> - 机器人 ID 不能与房间内的普通用户 ID 冲突，否则会导致转推任务由于机器人用户被踢出 TRTC 房间而异常结束。

旁路直播跨房连麦

1. 业务后台在每个旁路直播间启动一个旁路转推 CDN 任务 StartPublishCdnStream 。
2. 房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。
3. 房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
4. 业务后台同时启动 N 个混流回推房间任务 StartPublishCdnStream 。
    - 任务一：A_Agent 机器人接收 A 房间主播媒体流，经 TRTC 后台混流后由 A_Feed 机器人回推到 B 房间和 N 房间。
    - 任务二：B_Agent 机器人接收 B 房间主播媒体流，经 TRTC 后台混流后由 B_Feed 机器人回推到 A 房间和 N 房间。
    - 任务 N：N_Agent 机器人接收 N 房间主播媒体流，经 TRTC 后台混流后由 N_Feed 机器人回推到 A 房间和 B 房间。
5. 房间 A、房间 B、房间 N 的用户互相拉取房间中混流回推的音视频流，开始进行跨房 PK。
6. 业务后台更新参与跨房 PK 的房间中原有的旁路转推 CDN 任务 UpdatePublishCdnStream，混合其他房间回推的音视频流。
7. 跨房 PK 结束，业务后台通过 TaskId 停止 N 个混流回推房间任务 StopPublishCdnStream 。
8. 业务后台更新参与跨房 PK 的房间中原有的旁路转推 CDN 任务 UpdatePublishCdnStream，剔除其他房间回推的音视频流。

> 注意：
> - 根据转推目标的不同，旁路转推 CDN 对应参数 McuPublishCdnParam，回推 TRTC 房间对应参数 McuFeedBackRoomParams 。
> - 若直播间为单主播直播场景，则在启动转推任务时可选择单流旁路转推 SingleSubscribeParams，从而节省混流转码费用。

示例代码

下面以纯音频场景为例，展示跨房 PK 混流回推房间任务的参数体示例。

版权所有：腾讯云计算（北京）有限责任公司

任务一

```
{
  "SdkAppId": 1400000000,
  "RoomId": "A",
  "RoomIdType": 1,
  "AgentParams": {
    "UserId": "A_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
  "AudioParams": {
    "AudioEncode": {
      "Codec": 0,
      "SampleRate": 48000,
      "Channel": 2,
      "BitRate": 64
    }
  },
  "FeedBackRoomParams": [
    {
      "RoomId": "B",
      "RoomIdType": 1,
      "UserId": "A_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    },
    {
      "RoomId": "N",
      "RoomIdType": 1,
      "UserId": "A_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    }
  ]
}
```

任务二

```
{
  "SdkAppId": 1400000000,
  "RoomId": "B",
  "RoomIdType": 1,
```

```
  "AgentParams": {
    "UserId": "B_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
  "AudioParams": {
    "AudioEncode": {
      "Codec": 0,
      "SampleRate": 48000,
      "Channel": 2,
      "BitRate": 64
    }
  },
  "FeedBackRoomParams": [
    {
      "RoomId": "A",
      "RoomIdType": 1,
      "UserId": "B_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    },
    {
      "RoomId": "N",
      "RoomIdType": 1,
      "UserId": "B_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    }
  ]
}
```

任务 N

```
{
  "SdkAppId": 1400000000,
  "RoomId": "N",
  "RoomIdType": 1,
  "AgentParams": {
    "UserId": "N_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
```

```
    "AudioParams": {
      "AudioEncode": {
        "Codec": 0,
        "SampleRate": 48000,
        "Channel": 2,
        "BitRate": 64
      }
    },
    "FeedBackRoomParams": [
      {
        "RoomId": "A",
        "RoomIdType": 1,
        "UserId": "N_Feed",
        "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
      },
      {
        "RoomId": "B",
        "RoomIdType": 1,
        "UserId": "N_Feed",
        "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
      }
    ]
  }
```

注意：

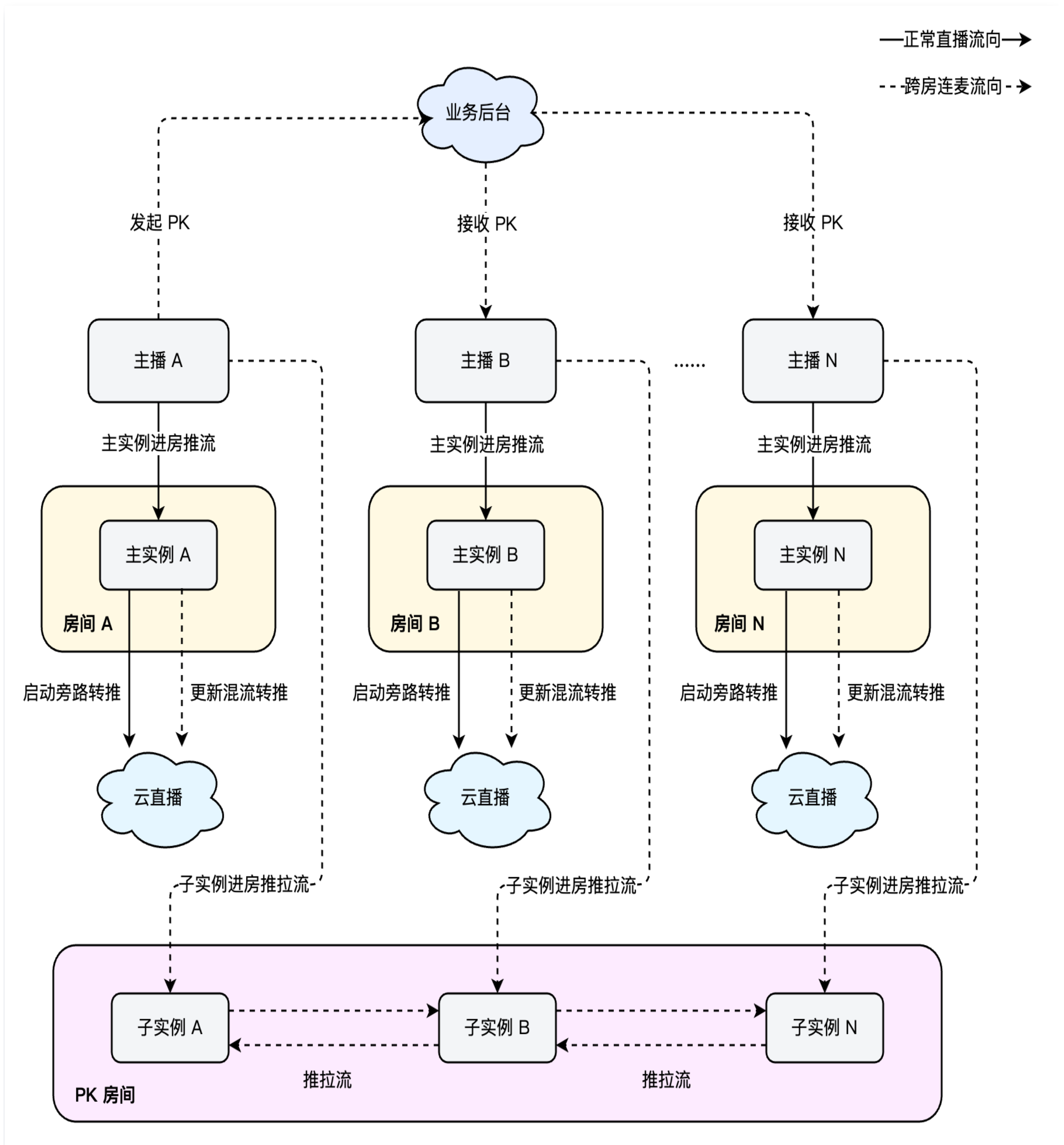
纯音频场景下，TRTC 后台会默认混合房间内所有主播音频流，亦可通过音频参数 McuAudioParams 指定音频混流黑白名单。

子实例跨房 PK 连麦方案

适用场景：多个房间 PK，每个房间有一个或多个主播的复杂跨房连麦场景。

方案原理

TRTC SDK 可以通过创建子实例的方式实现跨房 PK 连麦，此方案不限参与 PK 的房间数量，便于后期多个房间 PK 的业务拓展。开始跨房 PK 后，每个参与 PK 的主播均在本地创建一个子实例进入一个新的 PK 房间，这样不同房间的主播即可通过子实例推拉流实现音视频互通。



实现流程

实时互动跨房连麦

1. 房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。

2. 房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
3. 主播 A、主播 B、主播 N 分别创建一个子实例，进入一个新的 PK 房间进行推拉流。
4. 主播 A、主播 B、主播 N 的主实例分别启动一个混流回推房间任务 startPublishMediaStream，混合 PK 房间内所有子实例的音视频流并回推到自己房间，同时停止主实例推流。
5. 各房间观众用户拉取房间中混流回推的音视频流，各房间主播用户通过子实例拉取其他房间主播的音视频流，开始进行跨房 PK。
6. 跨房 PK 结束，各房间主播的主实例重启本地推流，并停止混流回推房间任务 stopPublishMediaStream，子实例退房并销毁。

旁路直播跨房连麦

1. 主播 A、主播 B、主播 N 的主实例分别启动一个旁路转推 CDN 任务 startPublishMediaStream。
2. 房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。
3. 房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
4. 主播 A、主播 B、主播 N 分别创建一个子实例，进入一个新的 PK 房间进行推拉流。
5. 主播 A、主播 B、主播 N 的主实例分别更新原有的旁路转推任务 updatePublishMediaStream，混合 PK 房间内所有子实例的音视频流并转推到直播 CDN，同时停止主实例推流。
6. 各房间观众用户拉取 CDN 流观看，各房间主播用户通过子实例拉取其他房间主播的音视频流，开始进行跨房 PK。
7. 跨房 PK 结束，各房间主播的主实例重启本地推流，并更新原混流转推任务 updatePublishMediaStream，恢复主实例单流旁路转推，子实例退房并销毁。

> **注意：**
> - 根据转推目标的不同，您可通过 TRTCPublishTarget 自行设置媒体流发布模式，目前支持单流转推、混流转码和回推房间模式。
> - 若涉及美颜特效的使用，建议主实例和子实例使用同一个美颜实例，设置第三方美颜的视频数据回调 时采用不同的 视频像素格式。

示例代码

1. 创建子实例，进入一个新的 PK 房间进行推拉流。

Android

```
// 初始化 TRTC 主实例
TRTCCloud mainCloud = TRTCCloud.sharedInstance(context);
// 创建 TRTC 子实例
TRTCCloud subCloud = mainCloud.createSubCloud();
// 添加子实例事件监听
```

```
subCloud.addListener(trtcSdkListener);

// 子实例进入 PK 房间
TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
params.sdkAppId = SDKAppId;
params.userId = "Sub_" + UserId;
params.userSig = UserSig;
params.role = TRTCCloudDef.TRTCRoleAnchor;
params.strRoomId = PK_RoomId;
subCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);

// 子实例开启本地音频采集和发布
subCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
// 子实例开启本地视频预览和发布
subCloud.startLocalPreview(mIsFrontCamera, mTxcvvAnchorPreviewView);

// 来自 TRTC SDK 的各类事件通知
private TRTCCloudListener trtcSdkListener = new TRTCCloudListener() {
    @Override
    public void onEnterRoom(long result) {
        if (result > 0) {
            // result 代表加入房间所消耗的时间（毫秒）
            Log.d(TAG, "Enter room succeed!");
        } else {
            // result 代表进房失败的错误码
            Log.d(TAG, "Enter room failed!");
        }
    }

    @Override
    public void onUserAudioAvailable(String userId, boolean available) {
        // 某远端用户发布/取消了自己的音频
        // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
    }

    @Override
    public void onUserVideoAvailable(String userId, boolean available) {
        // 某远端用户发布/取消了主路视频画面
        if (available) {
            // 订阅远端用户的视频流，并绑定视频渲染控件
            subCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, remoteView);
        } else {
```

```
            // 停止订阅远端用户的视频流，并释放渲染控件
            subCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
        }
    }
};
```

iOS

```
// 初始化 TRTC 主实例
TRTCCloud *mainCloud = [TRTCCloud sharedInstance];
// 创建 TRTC 子实例
TRTCCloud *subCloud = [mainCloud createSubCloud];
// 添加子实例事件监听
subCloud.delegate = self;

// 子实例进入 PK 房间
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = self.SDKAppId;
params.userId = [NSString stringWithFormat:@"Sub_%@", self.UserId];
params.userSig = self.UserSig;
params.role = TRTCRoleAnchor;
params.strRoomId = self.PK_RoomId;
[subCloud enterRoom:params appScene:TRTCAppSceneLIVE];

// 子实例开启本地音频采集和发布
[subCloud startLocalAudio:TRTCAudioQualityDefault];
// 子实例开启本地视频预览和发布
[subCloud startLocalPreview:self.isFrontCamera
view:self.anchorPreviewView];

// 来自 TRTC SDK 的各类事件通知
- (void)onEnterRoom:(NSInteger)result {
    if (result > 0) {
        // result 代表加入房间所消耗的时间(毫秒)
        [self toastTip:@"Enter room succeed!"];
    } else {
        // result 代表进房失败的错误码
        [self toastTip:@"Enter room failed!"];
    }
}
```

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了自己的音频
    // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了主路视频画面
    if (available) {
        // 订阅远端用户的视频流，并绑定视频渲染控件
        [subCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 停止订阅远端用户的视频流，并释放渲染控件
        [subCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
    }
}
```

2. （实时互动跨房连麦）启动混流回推房间任务，同时停止主实例推流。

Android

```
// 开始发布混合媒体流到 TRTC 房间
public void startPublishMediaToRoom(List<String> mixUserList) {
    // 媒体流发布的目标地址
    TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
    // 目标地址设定为混流回推到房间
    target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
    target.mixStreamIdentity.strRoomId = RoomId;
    // 混流机器人用户名不能与房间内其他用户重复
    target.mixStreamIdentity.userId = UserId + "_robot";

    // 设置媒体流编码输出参数
    TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
    trtcStreamEncoderParam.audioEncodedChannelNum = 1;
    trtcStreamEncoderParam.audioEncodedKbps = 50;
    trtcStreamEncoderParam.audioEncodedCodecType = 0;
    trtcStreamEncoderParam.audioEncodedSampleRate = 48000;
```

版权所有：腾讯云计算（北京）有限责任公司

```
    trtcStreamEncoderParam.videoEncodedFPS = 15;
    trtcStreamEncoderParam.videoEncodedGOP = 2;
    trtcStreamEncoderParam.videoEncodedKbps = 1300;
    trtcStreamEncoderParam.videoEncodedWidth = 540;
    trtcStreamEncoderParam.videoEncodedHeight = 960;

    // 媒体流转码配置参数
    TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new
TRTCCloudDef.TRTCStreamMixingConfig();
    if (mixUserList != null) {
        ArrayList<TRTCCloudDef.TRTCUser> audioMixUserList = new
ArrayList<>();
        ArrayList<TRTCCloudDef.TRTCVideoLayout> videoLayoutList = new
ArrayList<>();

        for (int i = 0; i < mixUserList.size() && i < 16; i++) {
            TRTCCloudDef.TRTCUser user = new TRTCCloudDef.TRTCUser();
            user.strRoomId = PK_RoomId;
            user.userId = mixUserList.get(i);
            audioMixUserList.add(user);

            TRTCCloudDef.TRTCVideoLayout videoLayout = new
TRTCCloudDef.TRTCVideoLayout();
            if (mixUserList.get(i).equals("Sub_" + UserId)) {
                // 本地主播画面布局
                videoLayout.x = 0;
                videoLayout.y = 0;
                videoLayout.width = 540;
                videoLayout.height = 960;
                videoLayout.zOrder = 0;
            } else {
                // PK 主播画面布局
                videoLayout.x = 400;
                videoLayout.y = 5 + i * 245;
                videoLayout.width = 135;
                videoLayout.height = 240;
                videoLayout.zOrder = 1;
            }
            videoLayout.fixedVideoUser = user;
            videoLayout.fixedVideoStreamType =
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG;
            videoLayoutList.add(videoLayout);
        }
```

版权所有：腾讯云计算（北京）有限责任公司

```
        // 指定转码流中的每一路输入音频的信息
        trtcStreamMixingConfig.audioMixUserList = audioMixUserList;
        // 指定混合画面的中每一路视频画面的位置、大小、图层以及流类型等信息
        trtcStreamMixingConfig.videoLayoutList = videoLayoutList;
    }

    // 开始发布媒体流
    mainCloud.startPublishMediaStream(target, trtcStreamEncoderParam,
trtcStreamMixingConfig);
}

// 开始发布媒体流的事件回调
@Override
public void onStartPublishMediaStream(String taskId, int code, String
message, Bundle extraInfo) {
    // taskId: 当请求成功时，TRTC 后台会在回调中提供给您这项任务的 taskId，后续您
可以通过该 taskId 结合 updatePublishMediaStream 和 stopPublishMediaStream
进行更新和停止
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 主实例停止推流
        mainCloud.stopLocalAudio();
        mainCloud.stopLocalPreview();
    }
}
```

iOS

```
// 开始发布混合媒体流到 TRTC 房间
- (void)startPublishMediaToRoom {
    // 媒体流发布的目标地址
    TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
    // 目标地址设定为混流回推到房间
    target.mode = TRTCPublishMixStreamToRoom;
    TRTCUser *mixStreamIdentity = [[TRTCUser alloc] init];
    mixStreamIdentity.strRoomId = self.RoomId;
    // 混流机器人用户名不能与房间内其他用户重复
    mixStreamIdentity.userId = [self.UserId
stringByAppendingString:@"_robot"];
    target.mixStreamIdentity = mixStreamIdentity;
```

```
    // 设置媒体流编码输出参数
    TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam
alloc] init];
    encoderParam.audioEncodedSampleRate = 48000;
    encoderParam.audioEncodedChannelNum = 1;
    encoderParam.audioEncodedKbps = 50;
    encoderParam.audioEncodedCodecType = 0;
    encoderParam.videoEncodedWidth = 540;
    encoderParam.videoEncodedHeight = 960;
    encoderParam.videoEncodedFPS = 15;
    encoderParam.videoEncodedGOP = 2;
    encoderParam.videoEncodedKbps = 1300;

    TRTCStreamMixingConfig *config = [[TRTCStreamMixingConfig alloc]
init];
    if (self.mixUserList.count) {
        NSMutableArray<TRTCUser *> *userList = [NSMutableArray array];
        NSMutableArray<TRTCVideoLayout *> *layoutList = [NSMutableArray
array];
        for (int i = 1; i < MIN(self.mixUserList.count, 16); i++) {
            TRTCUser *user = [[TRTCUser alloc] init];
            user.strRoomId = self.PK_RoomId;
            user.userId = self.mixUserList[i];
            [userList addObject:user];

            TRTCVideoLayout *layout = [[TRTCVideoLayout alloc] init];
            if ([self.mixUserList[i] isEqualToString:[NSString
stringWithFormat:@"Sub_%@", self.UserId]]) {
                // 本地主播画面布局
                layout.rect = CGRectMake(0, 0, 540, 960);
                layout.zOrder = 0;
            } else {
                // PK 主播画面布局
                layout.rect = CGRectMake(400, 5 + i * 245, 135, 240);
                layout.zOrder = 1;
            }
            layout.fixedVideoUser = user;
            layout.fixedVideoStreamType = TRTCVideoStreamTypeBig;
            [layoutList addObject:layout];
        }
        // 指定转码流中的每一路输入音频的信息
        config.audioMixUserList = [userList copy];
        // 指定混合画面的中每一路视频画面的位置、大小、图层以及流类型等信息
```

版权所有：腾讯云计算（北京）有限责任公司

```
        config.videoLayoutList = [layoutList copy];
    }
    // 开始发布媒体流
    [self.mainCloud startPublishMediaStream:target
encoderParam:encoderParam mixingConfig:config];
}

// 开始发布媒体流的事件回调
- (void)onStartPublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // taskId: 当请求成功时，TRTC 后台会在回调中提供给您这项任务的 taskId，后续您
可以通过该 taskId 结合 updatePublishMediaStream 和 stopPublishMediaStream
进行更新和停止
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 主实例停止推流
        [self.mainCloud stopLocalAudio];
        [self.mainCloud stopLocalPreview];
    }
}
```

3. （旁路直播跨房连麦）更新原有旁路转推任务，同时停止主实例推流。

Android

```
// 更新发布混合媒体流到直播 CDN
public void updatePublishMediaToCDN(String streamName, List<String>
mixUserList, String taskId) {
    // 设定推流地址过期时间
    long txTime = (System.currentTimeMillis() / 1000) + (24 * 60 * 60);
    // 生成鉴权信息，getSafeUrl 方法可在云直播控制台-域名管理-推流配置-推流地址示例
代码获取
    String secretParam = UrlHelper.getSafeUrl(LIVE_URL_KEY, streamName,
txTime);

    // 媒体流发布的目标地址
    TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
    // 目标地址设定为混流转推到 CDN
    target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToCdn;
    TRTCCloudDef.TRTCPublishCdnUrl cdnUrl = new
TRTCCloudDef.TRTCPublishCdnUrl();
```

```
    // 拼接发布到直播服务商的推流地址（RTMP 格式）
    cdnUrl.rtmpUrl = "rtmp://" + PUSH_DOMAIN + "/live/" + streamName +
"?" + secretParam;
    // 腾讯云直播服务为 true，第三方直播服务为 false
    cdnUrl.isInternalLine = true;
    // 可以添加多个 CDN 推流地址
    target.cdnUrlList.add(cdnUrl);

    // 设置媒体流编码输出参数
    TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
    trtcStreamEncoderParam.audioEncodedChannelNum = 1;
    trtcStreamEncoderParam.audioEncodedKbps = 50;
    trtcStreamEncoderParam.audioEncodedCodecType = 0;
    trtcStreamEncoderParam.audioEncodedSampleRate = 48000;
    trtcStreamEncoderParam.videoEncodedFPS = 15;
    trtcStreamEncoderParam.videoEncodedGOP = 2;
    trtcStreamEncoderParam.videoEncodedKbps = 1300;
    trtcStreamEncoderParam.videoEncodedWidth = 540;
    trtcStreamEncoderParam.videoEncodedHeight = 960;

    // 媒体流转码配置参数
    TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new
TRTCCloudDef.TRTCStreamMixingConfig();
    if (mixUserList != null) {
        ArrayList<TRTCCloudDef.TRTCUser> audioMixUserList = new
ArrayList<>();
        ArrayList<TRTCCloudDef.TRTCVideoLayout> videoLayoutList = new
ArrayList<>();

        for (int i = 0; i < mixUserList.size() && i < 16; i++) {
            TRTCCloudDef.TRTCUser user = new TRTCCloudDef.TRTCUser();
            user.strRoomId = PK_RoomId;
            user.userId = mixUserList.get(i);
            audioMixUserList.add(user);

            TRTCCloudDef.TRTCVideoLayout videoLayout = new
TRTCCloudDef.TRTCVideoLayout();
            if (mixUserList.get(i).equals("Sub_" + UserId)) {
                // 本地主播画面布局
                videoLayout.x = 0;
                videoLayout.y = 0;
                videoLayout.width = 540;
```

```
                videoLayout.height = 960;
                videoLayout.zOrder = 0;
            } else {
                // PK 主播画面布局
                videoLayout.x = 400;
                videoLayout.y = 5 + i * 245;
                videoLayout.width = 135;
                videoLayout.height = 240;
                videoLayout.zOrder = 1;
            }
            videoLayout.fixedVideoUser = user;
            videoLayout.fixedVideoStreamType =
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG;
            videoLayoutList.add(videoLayout);
        }

        // 指定转码流中的每一路输入音频的信息
        trtcStreamMixingConfig.audioMixUserList = audioMixUserList;
        // 指定混合画面的中每一路视频画面的位置、大小、图层以及流类型等信息
        trtcStreamMixingConfig.videoLayoutList = videoLayoutList;
    }

    // 更新发布媒体流
    mainCloud.updatePublishMediaStream(taskId, target,
trtcStreamEncoderParam, trtcStreamMixingConfig);
}

// 更新媒体流的事件回调
@Override
public void onUpdatePublishMediaStream(String taskId, int code, String
message, Bundle extraInfo) {
    // 您调用媒体流发布接口 (updatePublishMediaStream) 时传入的 taskId，会通过
此回调再带回给您，用于标识该回调属于哪一次更新请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 主实例停止推流
        mainCloud.stopLocalAudio();
        mainCloud.stopLocalPreview();
    }
}
```

iOS

```
// 更新发布混合媒体流到直播 CDN
- (void)updatePublishMediaToCDN {
    NSDate *date = [NSDate dateWithTimeIntervalSinceNow:0];
    // 设定推流地址过期时间
    NSTimeInterval time = [date timeIntervalSince1970] + (24 * 60 * 60);
    // 生成鉴权信息，getSafeUrl 方法可在云直播控制台-域名管理-推流配置-推流地址示例
代码获取
    NSString *secretParam = [self getSafeUrl:LIVE_URL_KEY
streamName:self.streamName time:time];

    // 媒体流发布的目标地址
    TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
    // 目标地址设定为混流转推到 CDN
    target.mode = TRTCPublishMixStreamToCdn;
    TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
    // 拼接发布到直播服务商的推流地址（RTMP 格式）
    cdnUrl.rtmpUrl = [NSString stringWithFormat:@"rtmp://%@/live/%@?%@",
PUSH_DOMAIN, self.streamName, secretParam];
    // 腾讯云直播推流地址为 true，第三方为 false
    cdnUrl.isInternalLine = YES;
    NSMutableArray* cdnUrlList = [NSMutableArray array];
    // 可以添加多个 CDN 推流地址
    [cdnUrlList addObject:cdnUrl];
    target.cdnUrlList = cdnUrlList;

    // 设置媒体流编码输出参数
    TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam
alloc] init];
    encoderParam.audioEncodedSampleRate = 48000;
    encoderParam.audioEncodedChannelNum = 1;
    encoderParam.audioEncodedKbps = 50;
    encoderParam.audioEncodedCodecType = 0;
    encoderParam.videoEncodedWidth = 540;
    encoderParam.videoEncodedHeight = 960;
    encoderParam.videoEncodedFPS = 15;
    encoderParam.videoEncodedGOP = 2;
    encoderParam.videoEncodedKbps = 1300;

    TRTCStreamMixingConfig *config = [[TRTCStreamMixingConfig alloc]
init];
    if (self.mixUserList.count) {
        NSMutableArray<TRTCUser *> *userList = [NSMutableArray array];
```

版权所有：腾讯云计算（北京）有限责任公司

```
        NSMutableArray<TRTCVideoLayout *> *layoutList = [NSMutableArray
array];
        for (int i = 1; i < MIN(self.mixUserList.count, 16); i++) {
            TRTCUser *user = [[TRTCUser alloc] init];
            user.strRoomId = self.PK_RoomId;
            user.userId = self.mixUserList[i];
            [userList addObject:user];

            TRTCVideoLayout *layout = [[TRTCVideoLayout alloc] init];
            if ([self.mixUserList[i] isEqualToString:[NSString
stringWithFormat:@"Sub_%@", self.UserId]]) {
                // 本地主播画面布局
                layout.rect = CGRectMake(0, 0, 540, 960);
                layout.zOrder = 0;
            } else {
                // PK 主播画面布局
                layout.rect = CGRectMake(400, 5 + i * 245, 135, 240);
                layout.zOrder = 1;
            }
            layout.fixedVideoUser = user;
            layout.fixedVideoStreamType = TRTCVideoStreamTypeBig;
            [layoutList addObject:layout];
        }
        // 指定转码流中的每一路输入音频的信息
        config.audioMixUserList = [userList copy];
        // 指定混合画面的中每一路视频画面的位置、大小、图层以及流类型等信息
        config.videoLayoutList = [layoutList copy];
    }
    // 更新发布媒体流
    [self.mainCloud updatePublishMediaStream:self.taskId
publishTarget:target encoderParam:encoderParam mixingConfig:config];
}

// 更新媒体流的事件回调
- (void)onUpdatePublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // 您调用媒体流发布接口 (updatePublishMediaStream) 时传入的 taskId，会通过
此回调再带回给您，用于标识该回调属于哪一次更新请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 主实例停止推流
        [self.mainCloud stopLocalAudio];
        [self.mainCloud stopLocalPreview];
```

```
    }
}
```

4. 各房间主播子实例互相拉取音视频流，开始跨房 PK。

Android

```
@Override
public void onUserAudioAvailable(String userId, boolean available) {
    // 某远端用户发布/取消了自己的音频
    // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
}

@Override
public void onUserVideoAvailable(String userId, boolean available) {
    // 某远端用户发布/取消了主路视频画面
    if (available) {
        // 订阅远端用户的视频流，并绑定视频渲染控件
        subCloud.startRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, remoteView);
    } else {
        // 停止订阅远端用户的视频流，并释放渲染控件
        subCloud.stopRemoteView(userId,
TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
    }
}
```

iOS

```
- (void)onUserAudioAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了自己的音频
    // 在自动订阅模式下，您无需做任何操作，SDK 会自动播放远端用户音频
}

- (void)onUserVideoAvailable:(NSString *)userId available:
(BOOL)available {
    // 某远端用户发布/取消了主路视频画面
    if (available) {
        // 订阅远端用户的视频流，并绑定视频渲染控件
```

```
        [self.subCloud startRemoteView:userId
streamType:TRTCVideoStreamTypeBig view:self.remoteView];
    } else {
        // 停止订阅远端用户的视频流，并释放渲染控件
        [self.subCloud stopRemoteView:userId
streamType:TRTCVideoStreamTypeBig];
    }
}
```

5. （实时互动跨房连麦）跨房 PK 结束，主实例重新推流，停止混流回推房间，子实例退房并销毁。

Android

```
// 主实例开启本地音频采集和发布
mainCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
// 主实例开启本地视频预览和发布
mainCloud.startLocalPreview(mIsFrontCamera, mTxcvvAnchorPreviewView);

// 主实例停止混流回推房间任务
mainCloud.stopPublishMediaStream(taskId);

// 停止媒体流的事件回调
@Override
public void onStopPublishMediaStream(String taskId, int code, String
message, Bundle extraInfo) {
    // 您调用停止发布媒体流 (stopPublishMediaStream) 时传入的 taskId，会通过此
回调再带回给您，用于标识该回调属于哪一次停止请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 子实例退房并销毁
        subCloud.stopLocalAudio();
        subCloud.stopLocalPreview();
        subCloud.exitRoom();
        mainCloud.destroySubCloud(subCloud);
    }
}
```

iOS

```
// 主实例开启本地音频采集和发布
[self.mainCloud startLocalAudio:TRTCAudioQualityDefault];
```

```
// 主实例开启本地视频预览和发布
[self.mainCloud startLocalPreview:self.isFrontCamera
view:self.anchorPreviewView];

// 主实例停止混流回推房间任务
[self.mainCloud stopPublishMediaStream:self.taskId];

// 停止发布媒体流的事件回调
- (void)onStopPublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // 您调用停止发布媒体流 (stopPublishMediaStream) 时传入的 taskId，会通过此
回调再带回给您，用于标识该回调属于哪一次停止请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 子实例退房并销毁
        [self.subCloud stopLocalAudio];
        [self.subCloud stopLocalPreview];
        [self.subCloud exitRoom];
        [self.mainCloud destroySubCloud:self.subCloud];
    }
}
```

6.（旁路直播跨房连麦）跨房 PK 结束，主实例重新推流，更新旁路转推任务，子实例退房并销毁。

Android

```
// 主实例开启本地音频采集和发布
mainCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_DEFAULT);
// 主实例开启本地视频预览和发布
mainCloud.startLocalPreview(mIsFrontCamera, mTxcvvAnchorPreviewView);

// 主实例更新旁路转推任务
public void updatePublishMediaToCDN(String streamName, String taskId) {
    // 设定推流地址过期时间
    long txTime = (System.currentTimeMillis() / 1000) + (24 * 60 * 60);
    // 生成鉴权信息，getSafeUrl 方法可在云直播控制台-域名管理-推流配置-推流地址示例
代码获取
    String secretParam = UrlHelper.getSafeUrl(LIVE_URL_KEY, streamName,
txTime);

    // 媒体流发布的目标地址
```

```
    TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
    // 目标地址设定为旁路转推到 CDN
    target.mode = TRTCCloudDef.TRTC_PublishBigStream_ToCdn;
    TRTCCloudDef.TRTCPublishCdnUrl cdnUrl = new
TRTCCloudDef.TRTCPublishCdnUrl();
    // 拼接发布到直播服务商的推流地址（RTMP 格式）
    cdnUrl.rtmpUrl = "rtmp://" + PUSH_DOMAIN + "/live/" + streamName +
"?" + secretParam;
    // 腾讯云直播服务为 true，第三方直播服务为 false
    cdnUrl.isInternalLine = true;
    // 可以添加多个 CDN 推流地址
    target.cdnUrlList.add(cdnUrl);

    // 设置媒体流编码输出参数
    TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
    trtcStreamEncoderParam.audioEncodedChannelNum = 1;
    trtcStreamEncoderParam.audioEncodedKbps = 50;
    trtcStreamEncoderParam.audioEncodedCodecType = 0;
    trtcStreamEncoderParam.audioEncodedSampleRate = 48000;
    trtcStreamEncoderParam.videoEncodedFPS = 15;
    trtcStreamEncoderParam.videoEncodedGOP = 2;
    trtcStreamEncoderParam.videoEncodedKbps = 1300;
    trtcStreamEncoderParam.videoEncodedWidth = 540;
    trtcStreamEncoderParam.videoEncodedHeight = 960;

    // 更新发布媒体流
    mainCloud.updatePublishMediaStream(taskId, target,
trtcStreamEncoderParam, null);
}

// 更新媒体流的事件回调
@Override
public void onUpdatePublishMediaStream(String taskId, int code, String
message, Bundle extraInfo) {
    // 您调用媒体流发布接口 (updatePublishMediaStream) 时传入的 taskId，会通过
此回调再带回给您，用于标识该回调属于哪一次更新请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 子实例退房并销毁
        subCloud.stopLocalAudio();
        subCloud.stopLocalPreview();
```

```
        subCloud.exitRoom();
        mainCloud.destroySubCloud(subCloud);
    }
}
```

iOS

```
// 主实例开启本地音频采集和发布
[self.mainCloud startLocalAudio:TRTCAudioQualityDefault];
// 主实例开启本地视频预览和发布
[self.mainCloud startLocalPreview:self.isFrontCamera
view:self.anchorPreviewView];

// 主实例更新旁路转推任务
- (void)updatePublishMediaToCDN {
    NSDate *date = [NSDate dateWithTimeIntervalSinceNow:0];
    // 设定推流地址过期时间
    NSTimeInterval time = [date timeIntervalSince1970] + (24 * 60 * 60);
    // 生成鉴权信息，getSafeUrl 方法可在云直播控制台-域名管理-推流配置-推流地址示例
代码获取
    NSString *secretParam = [self getSafeUrl:LIVE_URL_KEY
streamName:self.streamName time:time];

    // 媒体流发布的目标地址
    TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
    // 目标地址设定为旁路转推到 CDN
    target.mode = TRTCPublishBigStreamToCdn;
    TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
    // 拼接发布到直播服务商的推流地址（RTMP 格式）
    cdnUrl.rtmpUrl = [NSString stringWithFormat:@"rtmp://%@/live/%@?%@",
PUSH_DOMAIN, self.streamName, secretParam];
    // 腾讯云直播推流地址为 true，第三方为 false
    cdnUrl.isInternalLine = YES;
    NSMutableArray* cdnUrlList = [NSMutableArray array];
    // 可以添加多个 CDN 推流地址
    [cdnUrlList addObject:cdnUrl];
    target.cdnUrlList = cdnUrlList;

    // 设置媒体流编码输出参数
    TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam
alloc] init];
    encoderParam.audioEncodedSampleRate = 48000;
```

```
    encoderParam.audioEncodedChannelNum = 1;
    encoderParam.audioEncodedKbps = 50;
    encoderParam.audioEncodedCodecType = 0;
    encoderParam.videoEncodedWidth = 540;
    encoderParam.videoEncodedHeight = 960;
    encoderParam.videoEncodedFPS = 15;
    encoderParam.videoEncodedGOP = 2;
    encoderParam.videoEncodedKbps = 1300;

    // 更新发布媒体流
    [self.mainCloud updatePublishMediaStream:self.taskId
publishTarget:target encoderParam:encoderParam mixingConfig:nil];
}

// 更新媒体流的事件回调
- (void)onUpdatePublishMediaStream:(NSString *)taskId code:(int)code
message:(NSString *)message extraInfo:(NSDictionary *)extraInfo {
    // 您调用媒体流发布接口 (updatePublishMediaStream) 时传入的 taskId，会通过
此回调再带回给您，用于标识该回调属于哪一次更新请求
    // code: 回调结果，0 表示成功，其余值表示失败
    if (code == 0) {
        // 子实例退房并销毁
        [self.subCloud stopLocalAudio];
        [self.subCloud stopLocalPreview];
        [self.subCloud exitRoom];
        [self.mainCloud destroySubCloud:self.subCloud];
    }
}
```

跨房 PK 连麦方案对比分析

上面介绍了三种不同的跨房 PK 连麦实现方案，它们各自有不同的适用场景。下面分四个维度对不同的跨房连麦方案进行对比分析。

方案类型	方案优势	方案劣势	房间及人数限制	推荐的使用场景
常规跨房 PK 连麦方案	两人 PK 调用逻辑简单	多人 PK 调用逻辑复杂	单个主播最多和其他房间的 9 个主播跨房 PK	两个房间，单主播（双人）跨房 PK
服务端跨房 PK 连麦方案	纯服务端方案，客户端无需额外处理	存在额外的机器人推拉流及混流费用	最多支持 11 个房间同时进行跨房 PK，每个房间最多支持	多个房间，多主播（多人）跨房 PK，纯服务端管理

			16 个主播同时参与跨房 PK	
子实例跨房 PK 连麦方案	不限制跨房数量，便于后期业务扩展	实现逻辑复杂，多实例的管理易出错	无限制	多个房间，单主播跨房 PK，后期业务可能扩展更多房间

版权所有：腾讯云计算（北京）有限责任公司

系统来电监听与处理

最近更新时间：2024-07-30 08:55:11

在使用语聊直播 App 的过程当中，经常会遇到移动设备系统来电的情形。此时会出现语聊直播 App 的音频采集被打断、音频播放干扰来电通话声音等问题。下面我们将分别介绍在 Android、iOS、Flutter 平台上，针对系统来电状态的监听方法，以及对应状态下 RTC 的处理策略。

Android

步骤一：声明权限

在 AndroidManifest.xml 文件中添加以下权限声明，以便应用能够访问电话状态。

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

步骤二：检查并申请权限

检查权限授予情况，并在运行时动态申请 READ_PHONE_STATE 权限。

```
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_PHONE_STATE)
        != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.READ_PHONE_STATE},
PERMISSION_REQUEST_CODE);
}
```

步骤三：注册广播接收器

注册广播接收器（BroadcastReceiver），用于监听特定的系统事件（如电话状态变化）。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // 注册广播接收器
    IntentFilter filter = new IntentFilter();
    filter.addAction("android.intent.action.PHONE_STATE");
    registerReceiver(callStateReceiver, filter);
}
```

```
@Override
protected void onDestroy() {
    super.onDestroy();

    // 取消注册广播接收器
    unregisterReceiver(callStateReceiver);
}
```

步骤四：接收电话状态变化

通过广播接收器（BroadcastReceiver）接收系统电话状态变化，并进行通话状态映射。

```
private final BroadcastReceiver callStateReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if
(intent.getAction().equals("android.intent.action.PHONE_STATE")) {
            String state =
intent.getStringExtra(TelephonyManager.EXTRA_STATE);
            if (TelephonyManager.EXTRA_STATE_IDLE.equals(state)) {
                // 空闲状态
                handleCallState("idle");
            }
            if (TelephonyManager.EXTRA_STATE_RINGING.equals(state)) {
                // 来电状态
                handleCallState("incoming");
            }
            if (TelephonyManager.EXTRA_STATE_OFFHOOK.equals(state)) {
                // 通话状态
                handleCallState("offhook");
            }
        }
    }
};
```

步骤五：处理电话状态变化

处理通话状态变化，在特定状态下进行 RTC 音频处理，避免对来电通话声音产生干扰。

```
private void handleCallState(String state) {
    Log.d("CallState", state);
```

```
    TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(this);
    if (!state.equals("idle")) {
        // 暂停播放所有远端用户的音频流
        mTRTCCloud.muteAllRemoteAudio(true);
    } else {
        // 恢复播放所有远端用户的音频流
        mTRTCCloud.muteAllRemoteAudio(false);
    }
}
```

iOS

iOS 平台有两类 API 可以监听获取系统电话状态，下面分别介绍两种方法的具体实现步骤。

方法一：CoreTelephony

这是一种较老的 API，虽然 Xcode 会提示该 API 已过时，但目前在 iOS 17 及以下系统版本仍能正常使用。

步骤一：引入头文件，声明 property

```
#import <CoreTelephony/CTCall.h>
#import <CoreTelephony/CTCallCenter.h>

@interface ViewController ()

@property (nonatomic, strong) CTCallCenter *callCenter;

@end
```

步骤二：注册回调，处理电话状态变化

```
self.callCenter = [[CTCallCenter alloc] init];

self.callCenter.callEventHandler = ^(CTCall *call) {
    if ([call.callState isEqualToString:CTCallStateDialing]) {
        NSLog(@"正在拨号");
        // 暂停播放所有远端用户的音频流
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
    else if([call.callState isEqualToString:CTCallStateIncoming]) {
        NSLog(@"来电");
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
```

```
    else if([call.callState isEqualToString:CTCallStateConnected]) {
        NSLog(@"正在通话");
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
    else if([call.callState isEqualToString:CTCallStateDisconnected]) {
        NSLog(@"断开通话");
        // 恢复播放所有远端用户的音频流
        [self.trtcCloud muteAllRemoteAudio:NO];
    }
    else {
        NSLog(@"未知");
    }
};
```

方法二：CallKit

这是一种新的 API，但目前在中国大陆地区使用 CallKit，可能会无法通过 App Store 的审核，请酌情谨慎选择。

步骤一：引入头文件，声明 property

```
#import <CallKit/CXCallObserver.h>
#import <CallKit/CXCall.h>

@interface ViewController () <CXCallObserverDelegate>

@property (nonatomic, strong) CXCallObserver *callObserver;

@end
```

步骤二：设置代理

```
self.callObserver = [[CXCallObserver alloc] init];
[self.callObserver setDelegate:self queue:dispatch_get_main_queue()];
```

步骤三：实现回调，监听通话状态

```
#pragma mark - CXCallObserverDelegate

- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {
    NSLog(@"The unique identifier for the call: %@", call.UUID);
```

```
    NSLog(@"outgoing(拨打):%d  onHold(挂起):%d  hasConnected(已接通):%d
hasEnded(已挂断):%d", call.outgoing, call.onHold, call.hasConnected,
call.hasEnded);

    // 来电
    if (!call.outgoing && !call.hasConnected && !call.hasEnded) {
        NSLog(@"来电");
        // 暂停播放所有远端用户的音频流
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
    // 拨打
    if (call.outgoing && !call.hasConnected && !call.hasEnded) {
        NSLog(@"拨打");
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
    // 接通
    if (call.hasConnected && !call.hasEnded) {
        NSLog(@"接通");
        [self.trtcCloud muteAllRemoteAudio:YES];
    }
    // 挂断
    if (call.hasEnded) {
        NSLog(@"挂断");
        // 恢复播放所有远端用户的音频流
        [self.trtcCloud muteAllRemoteAudio:NO];
    }
}
```

Flutter

Flutter 应用可以通过底层原生平台监听系统电话状态，然后利用 EventChannel 传递来自原生平台的事件。

步骤一：事件监听

在原生平台上，通过 StreamHandler 监听事件源（如系统电话状态变化）。

Android

```
private static final String CALL_STATUS_CHANNEL =
"com.example.call_status_listener/callStatus";
private EventChannel.EventSink eventSink;
```

```
new
EventChannel(getFlutterEngine().getDartExecutor().getBinaryMessenger(),
CALL_STATUS_CHANNEL).setStreamHandler(
        new EventChannel.StreamHandler() {
            @Override
            public void onListen(Object arguments,
EventChannel.EventSink events) {
                eventSink = events;
                IntentFilter filter = new IntentFilter();
                filter.addAction("android.intent.action.PHONE_STATE");
                registerReceiver(callStateReceiver, filter);
            }

            @Override
            public void onCancel(Object arguments) {
                eventSink = null;
                unregisterReceiver(callStateReceiver);
            }
        }
);
```

iOS

```
var eventSink: FlutterEventSink?
// 以CallKit为例，也可参考iOS原生部分的示例，使用CoreTelephony
let callObserver = CXCallObserver()

override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?
) -> Bool {
    GeneratedPluginRegistrant.register(with: self)
    let controller : FlutterViewController = window?.rootViewController
as! FlutterViewController
    let eventChannel = FlutterEventChannel(name:
"com.example.call_status_listener/callStatus",
                                binaryMessenger:
controller.binaryMessenger)
    eventChannel.setStreamHandler(self)
    callObserver.setDelegate(self, queue: nil)
```

```
    return super.application(application, didFinishLaunchingWithOptions:
launchOptions)
}

func onListen(withArguments arguments: Any?, eventSink events: @escaping
FlutterEventSink) -> FlutterError? {
    self.eventSink = events
    return nil
}

func onCancel(withArguments arguments: Any?) -> FlutterError? {
    self.eventSink = nil
    return nil
}
```

步骤二：事件发送

在原生平台上，当事件发生时，通过 EventSink 将事件发送到 Flutter。

Android

```
private final BroadcastReceiver callStateReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if
(intent.getAction().equals("android.intent.action.PHONE_STATE")) {
            String state =
intent.getStringExtra(TelephonyManager.EXTRA_STATE);
            if (eventSink != null) {
                if (TelephonyManager.EXTRA_STATE_IDLE.equals(state)) {
                    eventSink.success("idle");
                }
                if (TelephonyManager.EXTRA_STATE_RINGING.equals(state))
{
                    eventSink.success("incoming");
                }
                if (TelephonyManager.EXTRA_STATE_OFFHOOK.equals(state))
{
                    eventSink.success("offhook");
                }
            }
        }
```

```
    }
};
```

iOS

```
@objc func callObserver(_ callObserver: CXCallObserver, callChanged
call: CXCall) {
    // 挂断
    if call.hasEnded == true {
      eventSink?("idle")
    }
    // 拨打
    else if call.isOutgoing == true && call.hasConnected == false {
      eventSink?("offhook")
    }
    // 来电
    else if call.isOutgoing == false && call.hasConnected == false &&
call.hasEnded == false {
      eventSink?("incoming")
    }
    // 接通
    else if call.hasConnected == true && call.hasEnded == false {
      eventSink?("offhook")
    }
}
```

步骤三：事件接收

在 Flutter 端，通过 EventChannel 通道接收特定事件，并处理这些事件。

```
static const EventChannel _callStateChannel =
EventChannel("com.example.call_status_listener/callStatus");
String _callState = "Unknown";
TRTCCloud trtcCloud = (await TRTCCloud.sharedInstance())!;

_callStateChannel.receiveBroadcastStream().listen(_onCallStateChanged,
onError: _onError);

void _onCallStateChanged(Object? state) {
  setState(() {
    _callState = state.toString();
    print("_callState: " + _callState);
```

```
    if (_callState != "idle") {
      // 暂停播放所有远端用户的音频流
      trtcCloud.muteAllRemoteAudio(true);
    } else {
      // 恢复播放所有远端用户的音频流
      trtcCloud.muteAllRemoteAudio(false);
    }
  });
}

void _onError(Object error) {
  setState(() {
    _callState = "Failed to get call state: $error";
    print("_callState: " + _callState);
  });
}
```

客户端合流方案

最近更新时间：2024-04-12 11:41:11

应用场景

班课场景

在班课场景中，学员通常需要同时观看课件内容和老师视频画面。课件内容一般是 PPT、Word 等第三方窗口，在上课时，课件一般会全屏显示或者最大化显示，采集的时候可以基于窗口句柄采集，也可以采集整个屏幕画面。
除了使用后台混流方案，还可将本地摄像头的画面悬浮在屏幕的最顶层，然后对整个屏幕区域进行采集。这种方法的界面处理逻辑需要业务方自己开发实现，实现较为复杂，同时也因为采集的是整个屏幕区域，容易把其他程序的窗口误采集进去。误采集的窗口，可能会泄露老师隐私，同时也会使学员的注意力分散，影响上课效果。
相比悬浮视频窗口的方案，使用客户端本地合流方案会更好。采集课件窗口不采集整个屏幕区域，同时采集摄像头视频不用悬浮显示，这样就不会遮挡老师的课件区域，通过 TRTC SDK 在本地把课件内容和摄像头视频合为一路视频流。

会议直播场景

在会议直播场景中，存在多个摄像机机位一起直播的场景。业务方可以根据需要展示的机位数量开发不同的布局模板，例如左右并列、九宫格、一大多小等布局。
除了使用后台混流方案可以实现，还可使用客户端合流方案。客户端通过采集卡等外部设备，获得摄像机的视频流，加上 PPT 分享窗口，以及会议主办方 Logo 图片等，在本地合成一路视频流，再推流到 TRTC 后台。
如果中途需要切换摄像机画面，又不想改变布局时，则可以重新构造本地混流参数，修改视频源更新混流参数即可。

游戏直播场景

在 PC 端游戏直播场景中，除了游戏操作画面需要直播，还需要加上主播自己的摄像头画面。后台混流方案的实现是推两路流，一路桌面分享流、一路摄像头流，然后调用后台混流接口合为一路视频流。如果本地电脑的配置高，则可以使用客户端合流方案，在本地电脑把桌面分享流和摄像头流合为一路视频流，再推流到 TRTC 后台。

版权所有：腾讯云计算（北京）有限责任公司

后台混流方案和客户端合流方案对比

以上三个场景用后台混流方案和客户端合流方案均能实现，各有优势，详情如下：

后台混流方案

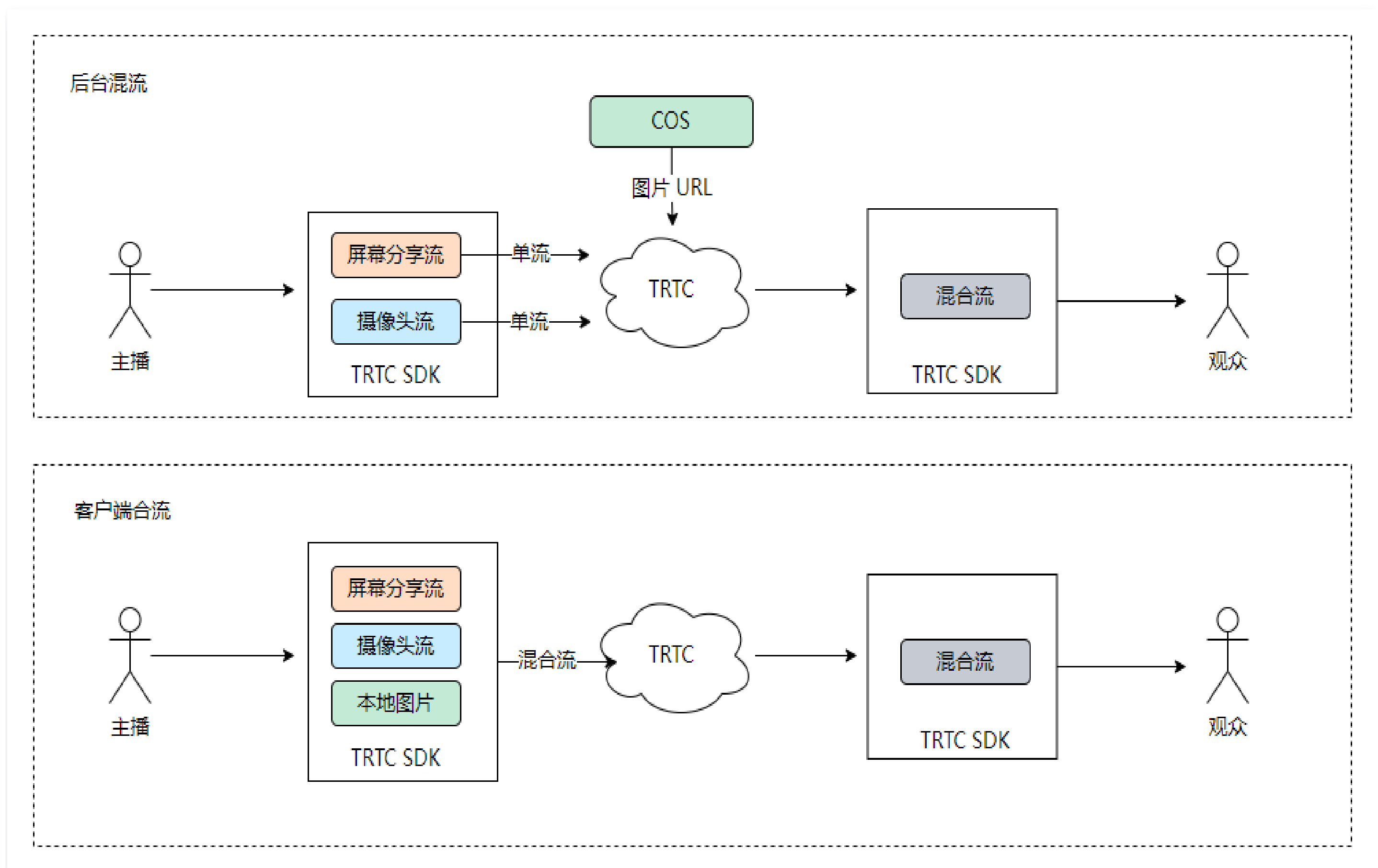
- 后台可以针对单流录制。
- 可以对单流进行二次处理，例如 AI 语音识别等。
- 对客户端配置要求不高。

客户端合流方案

- 业务方可以做出各种灵活的合流模板供主播自己选择。
- 费用低，没有后台混流成本。
- 对客户端电脑配置要求比较高。

后台混流方案与客户端合流方案的架构对比：

版权所有：腾讯云计算（北京）有限责任公司



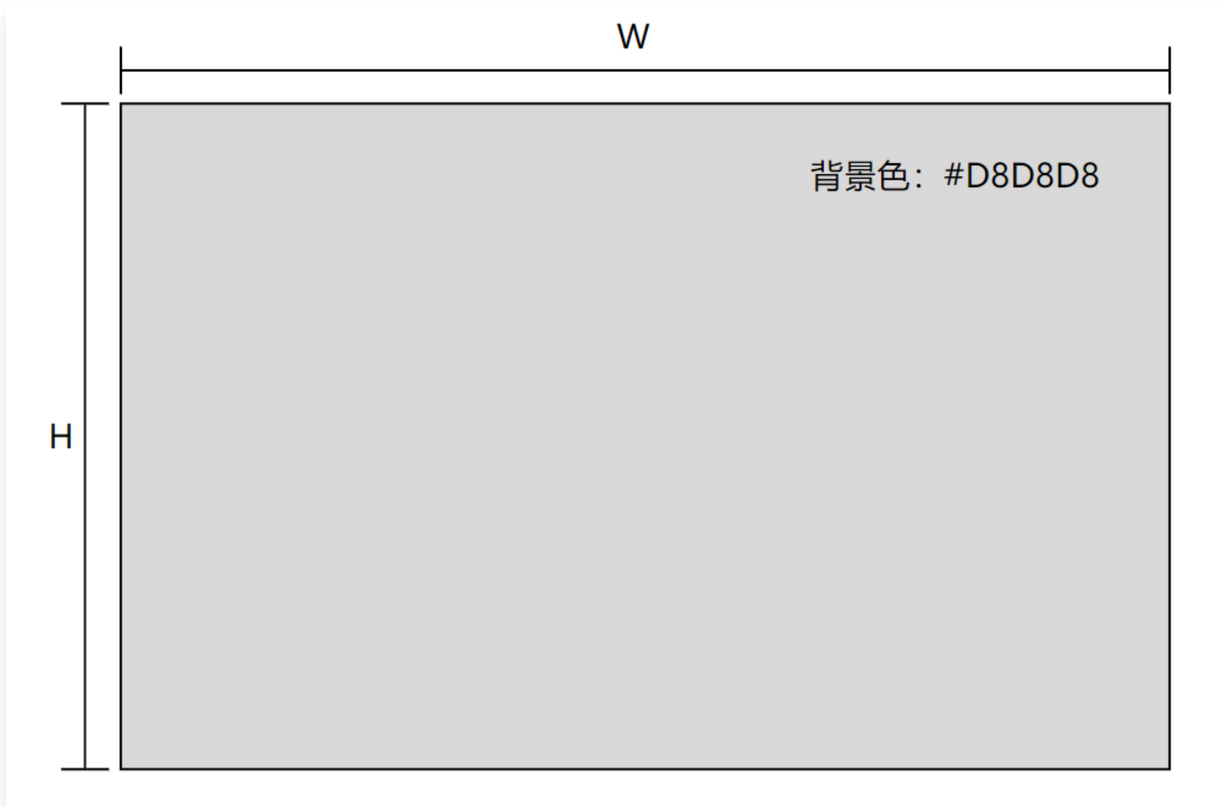
方案原理

合流的处理过程，可以理解为视频流的输出画面最下方有一层“幕布”，在“幕布”上一层叠加子画面。

输出画面的宽高、背景色、码率这几个参数需要特别注意：

- **画面宽高选择：**输出画面通常可以使用横屏模式，宽高比例可以选择 16:9，这个比例在手机和 PC 上播放的效果都不错。需要注意的是，如果严格按照16:9 的比例，有可能跟 PC 采集端屏幕分辨率比例不一致，导致“幕布”左右两边留有空白，这时则需要业务方进行处理，用纯色或者图片填充。还可以使用 PC 采集端的屏幕分辨率作为输出画面的宽高，这样屏幕层可以把整个“幕布”占满。
- **背景色的选择：**当子画面铺不满“幕布”时，可以设置输出画面的背景色。背景色可以根据具体业务需求设定，例如可以设置为播放器窗口背景色一样的颜色，在播放器播放视频时，视频铺不满播放器窗口的地方，视频背景色和播放器背景色融合在一起，看不出边界。
- **混流输出码率的选择：**可以根据分辨率、帧率设置对应的输出码率，输出过大会浪费带宽，输出过小则会导致视频模糊，具体可以参考 码率设置 。

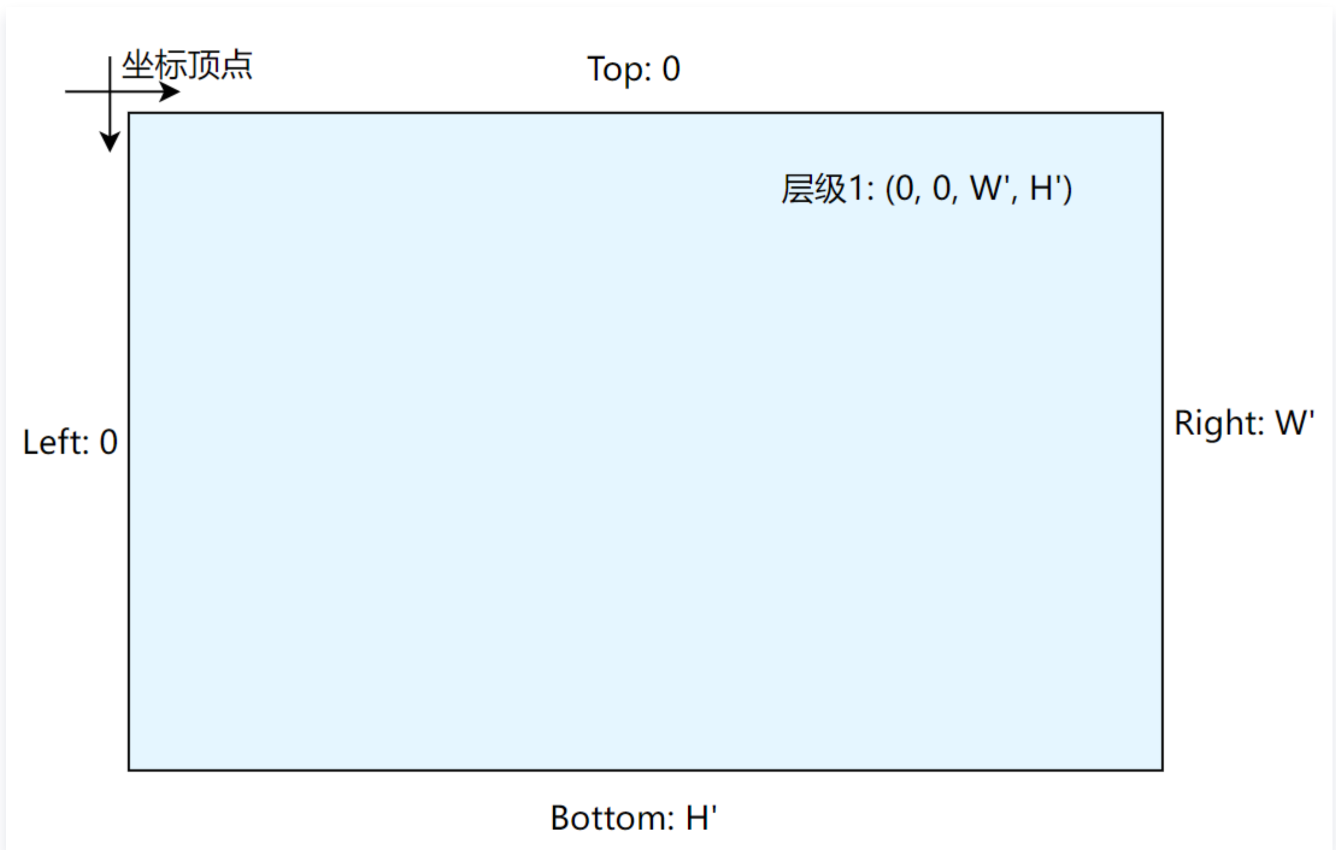
1. 设置输出画面的背景色和宽高，这里的背景色只是参考，具体选择根据业务需求决定。



2. 设置第一层子画面，通常是屏幕采集到的画面，例如整个游戏的画面。

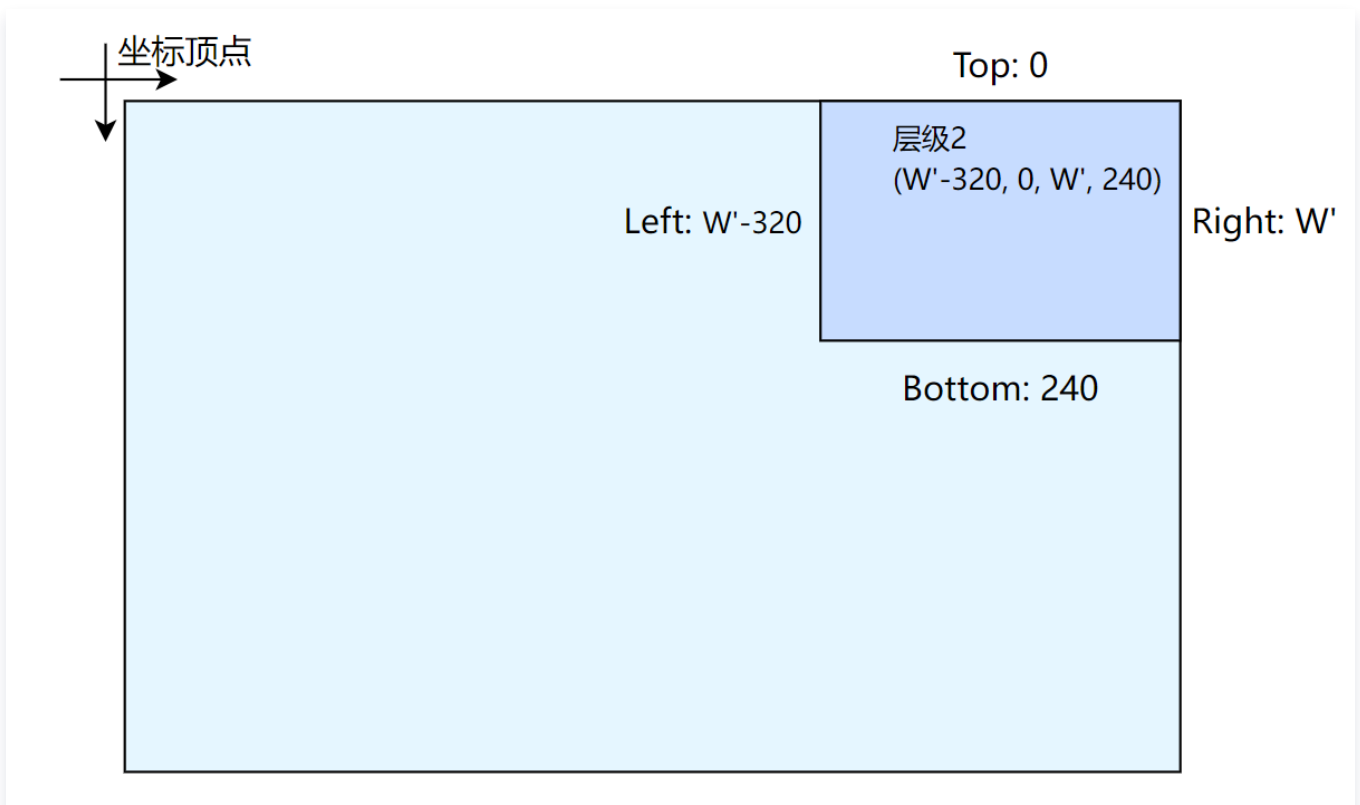
   第一层子画面的层级为一，坐标点的参数为 `RECT{left,top,right,bottom}`，坐标原点相对于“幕布”的左上角。如果第一层子画面跟“幕布”大小一致，则可以跟“幕布”的坐标一致。如果不一致，可以适应最短边，长边按比例拉伸后居中。

版权所有：腾讯云计算（北京）有限责任公司

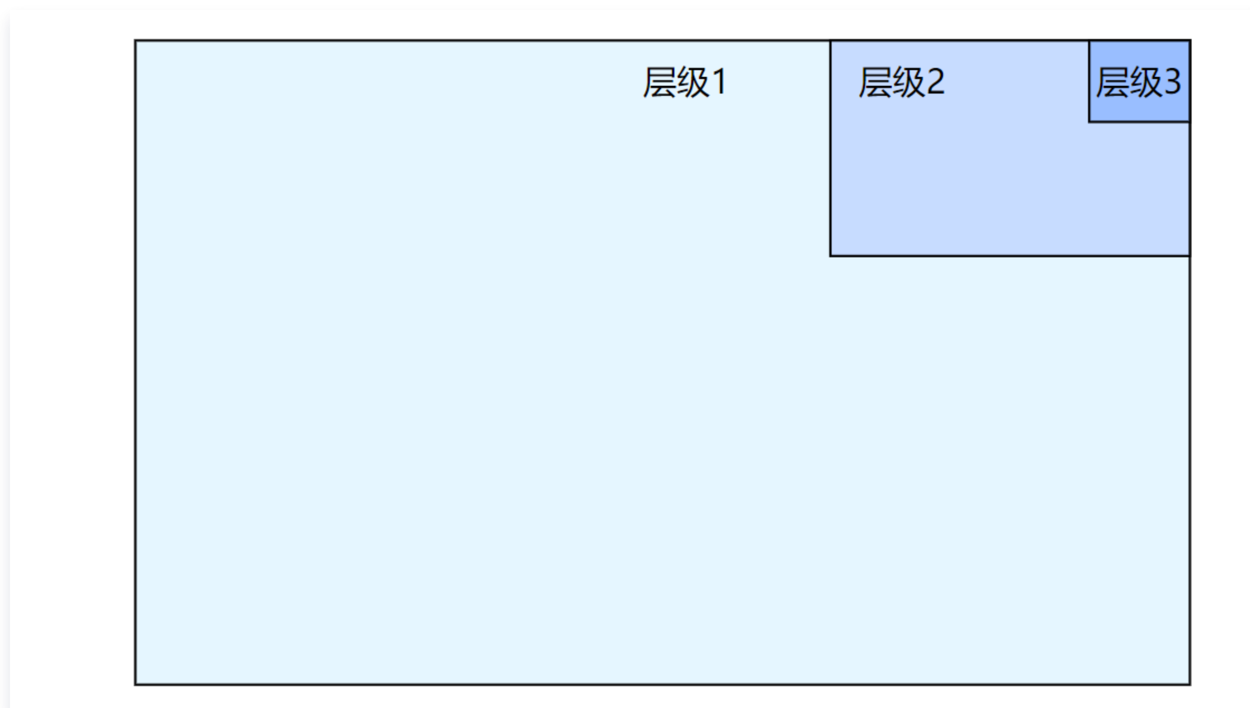


3. 设置第二层子画面，通常是主播自己的摄像头采集画面。

   为了提升体验，业务方可以在客户端实现一个子画面拖动摆放位置的功能，让主播自己调整子画面的大小和位置，这样就不会挡住上一层画面的重要内容。



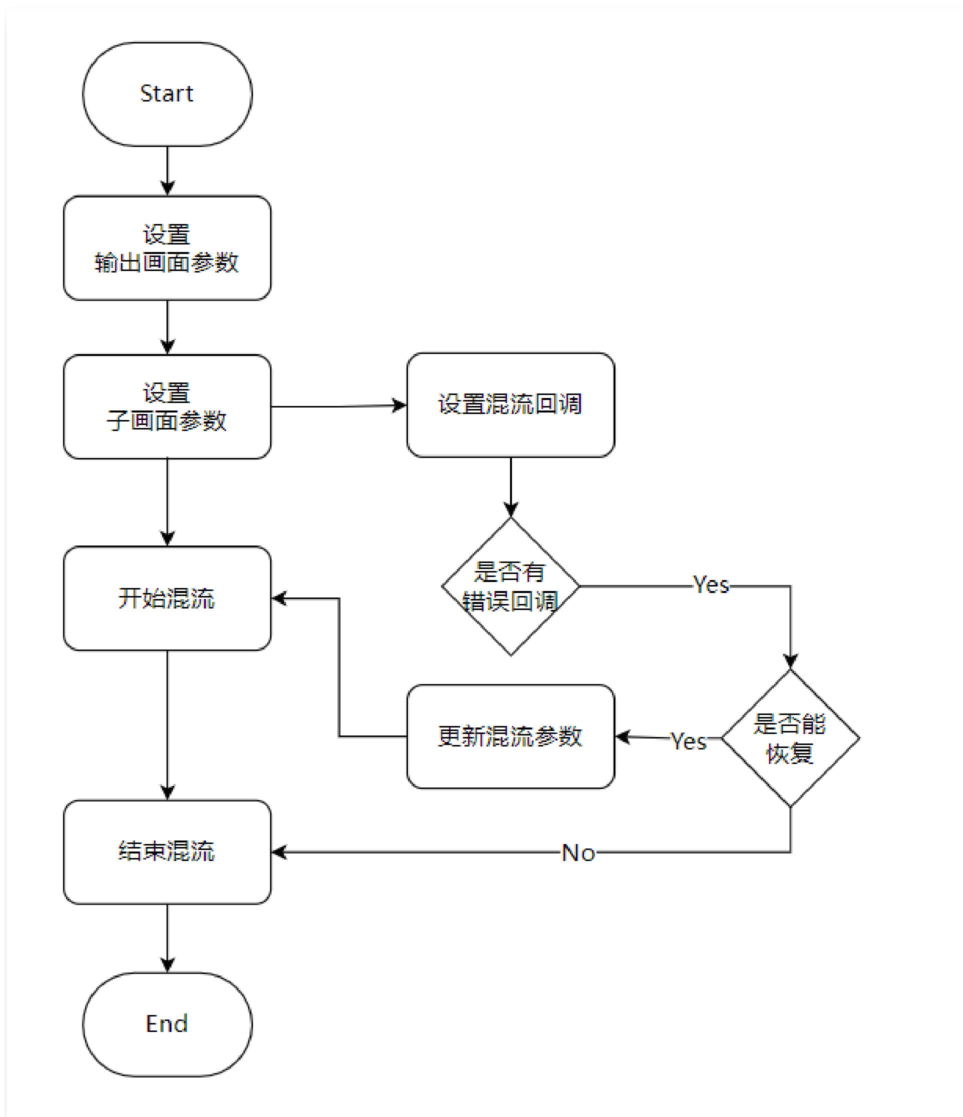
4. 设置第三层子画面，通常可以加个业务方 Logo 图片，可以设置一张静态图片或者动态图片。第三层子画面的位置不一定要叠在第二层的位置上，二层和三层表示层级关系，没有从属关系。



> **注意：**
> 这里的图片地址是本地图片的绝对路径，如果是网络地址图片，需要业务方提前下载到本地。

实现逻辑

客户端合流及推流流程



实现步骤

1. 设置输出画面的分辨率 `videoEncoderParams.videoResolution` 、帧率 `videoEncoderParams.videoFps` 和背景色 `canvasColor` 。
2. 设置第一层子画面的类型 `sourceType` 为 `MediaSourceScreen` ，设置需要采集的屏幕 `screenSourceId` 、子画面区域 `rect` 和子画面层级 `zOrder` 。
3. 设置第二层子画面的类型 `sourceType` 为 `MediaSourceCamera` ，设置需要采集的摄像头 `cameraDeviceId` 、子画面区域 `rect` 和子画面层级 `zOrder` 。
4. 设置第三层子画面的类型 `sourceType` 为 `MediaSourceImage` ，设置图片的本地路径 `imagePath` 、子画面区域 `rect` 和子画面层级 `zOrder` 。
5. 计算所有子画面的层级数量，设置输出画面的总层级数 `inputSourceListSize` 和所有层级信息 `inputSourceList` 。
6. 设置 SDK 合流状态和错误码回调 `setTranscodingCallback` 。
7. 开始本地合流并推流 `startTranscoding` 。

版权所有：腾讯云计算（北京）有限责任公司

8. 中途如果需要变更布局可以使用 `updateTranscodingParams` 。
9. 结束本地合流和推流 `stopTranscoding` 。

示例代码

1. 引入 ITXLocalMediaTranscoding.h 头文件，获取 `ITXLocalMediaTranscoding` 实例。

```
#include "ITXLocalMediaTranscoding.h"

// 创建合流实例
tx_local_media_transcoding_ = createTXLocalMediaTranscoding();
```

2. 设置输出画面相关参数：大小、颜色等。

```
auto canvas = std::make_shared<LocalMediaTranscodingParams>();

// 将输出画面大小设置为 1080P
TRTCVideoEncParam enc_params;
enc_params.videoResolution = TRTCVideoResolution_1920_1080;
enc_params.videoFps = 30;
canvas->videoEncoderParams = enc_params;

// 设置输出画面默认颜色
canvas->canvasColor = 0xFF0000;
```

3. 给输出画面添加源。

```
std::vector<LocalMediaTranscodingSource> sources;
TRTCRenderParams render_params;
// 设置渲染模式
render_params.fillMode = TRTCVideoFillMode_Fill;

// 添加屏幕分享源
LocalMediaTranscodingSource screen;

// 先通过 TRTC 接口获取到可分享的窗口列表
auto screen_capture_list = getTRTCShareInstance()-
>getScreenCaptureSources(thumb_size, icon_size);

// 添加第一个源
screen.screenSourceId = screen_capture_list->getSourceInfo(0).sourceId;
screen.rect = {0, 0, 1920, 1080};
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 目前支持 摄像头源、屏幕分享源、图片源
screen.sourceType = MediaSourceScreen;
screen.zOrder = 0;
sources.push_back(screen);
```

4. 绑定 TRTC 推流。

```
// 绑定 TRTC，需要在 TRTC 进房后绑定
tx_local_media_transcoding_->attachTRTC(getTRTCShareInstance());

// 开启采集
// 注意这里跟 TRTC 的 startScreenCapture 是互斥的，不能同时使用
// 摄像头也一样互斥
tx_local_media_transcoding_->startScreenSource(screen_capture_list-
>getSourceInfo(0),{0, 0, 0, 0});

// 开启合流， 并推流
tx_local_media_transcoding_->startTranscoding(TRTCVideoStreamTypeBig,
*sources);
```

版权所有：腾讯云计算（北京）有限责任公司