实时音视频

最佳实践

文档版本	01
发布日期	2024-11-20

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 云端录制与回放

1.1 概述

华为云实时音视频服务提供的云端录制回放功能适用于需要将音视频通话或互动直播过程进行录制和存储的业务场景。云端录制功能包含两种模式，如表1-1所示。

表 1-1 云端录制模式

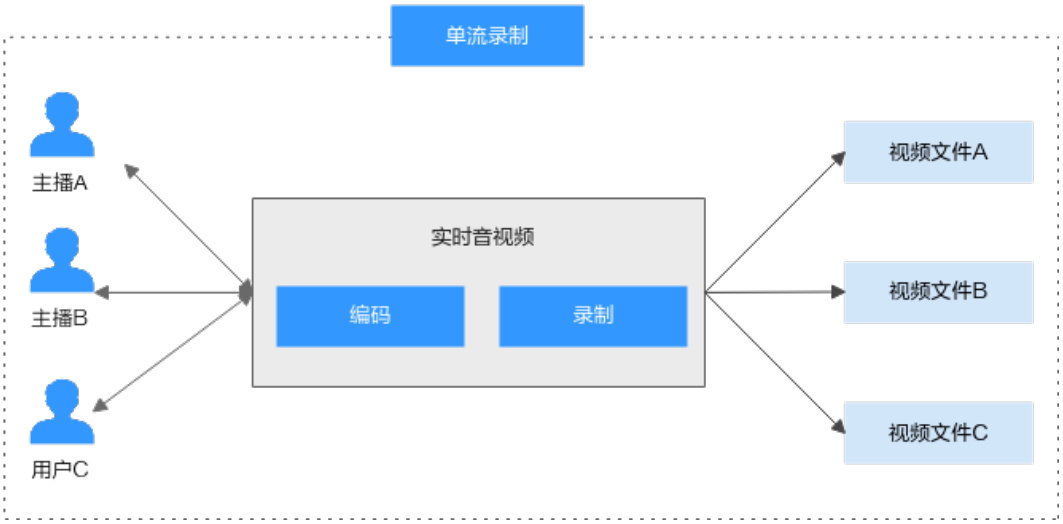
模式	说明	适用场景
单流录制	• 支持单流录制，可按用户进行录制。 • 支持自启动录制，开启后对房间内每一个流进行录制。 • 录制指定的媒体类型，支持仅录制音频、仅录制视频、同时录制音视频。 • 支持录制MP4、HLS文件，音频编码类型支持AAC。 • 支持获取回调消息中的播放URL进行回放，回调消息中的downloadurl字段为OBS播放地址，使用该URL播放将会在OBS中产生对应的下载流量或者带宽费用。 • 支持选择摄像头流或屏幕分享流。 • 支持指定分辨率大小。	在线课堂、内容审核等

模式	说明	适用场景
合流录制	• 支持合流录制，可多路视频或多路音频、视频合流录制。 • 录制指定的媒体类型，支持仅录制音频、仅录制视频、同时录制音视频。 • 支持录制MP4、HLS文件，音频编码类型支持AAC。 • 设置音视频属性，支持设置音视频属性，如码率、分辨率、帧率等。 • 支持获取回调消息中的播放URL进行回放，回调消息中的downloadurl字段为OBS播放地址，使用该URL播放将会在OBS中产生对应的下载流量或者带宽费用。	连麦直播等

1.2 单流录制

场景说明

将房间中的每一个用户的音视频流分别录制成独立的文件。



录制机制

SparkRTC提供的**单流录制**支持自启动模式，即**单流自动录制**，具体的实现机制如下图所示。

图 1-1 单流录制

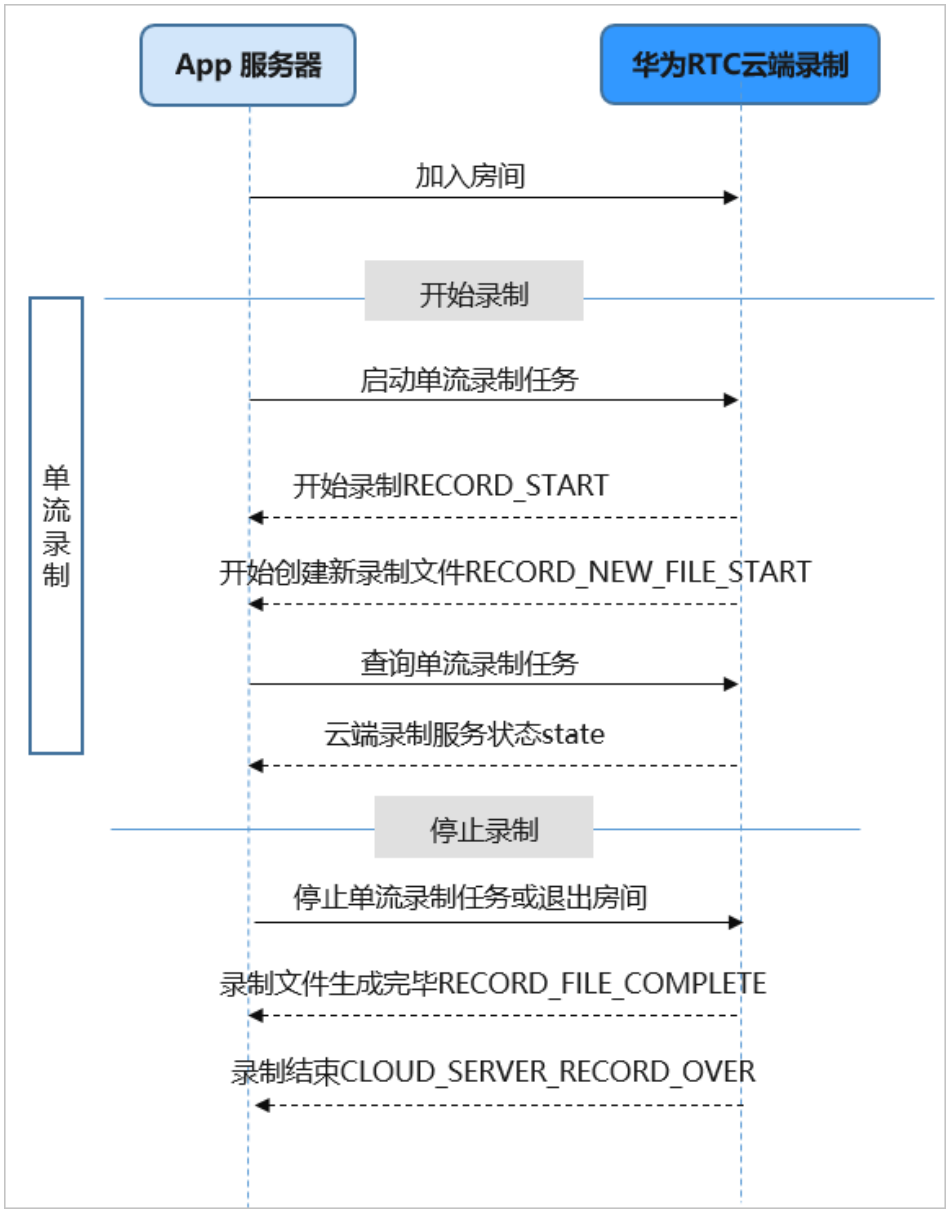
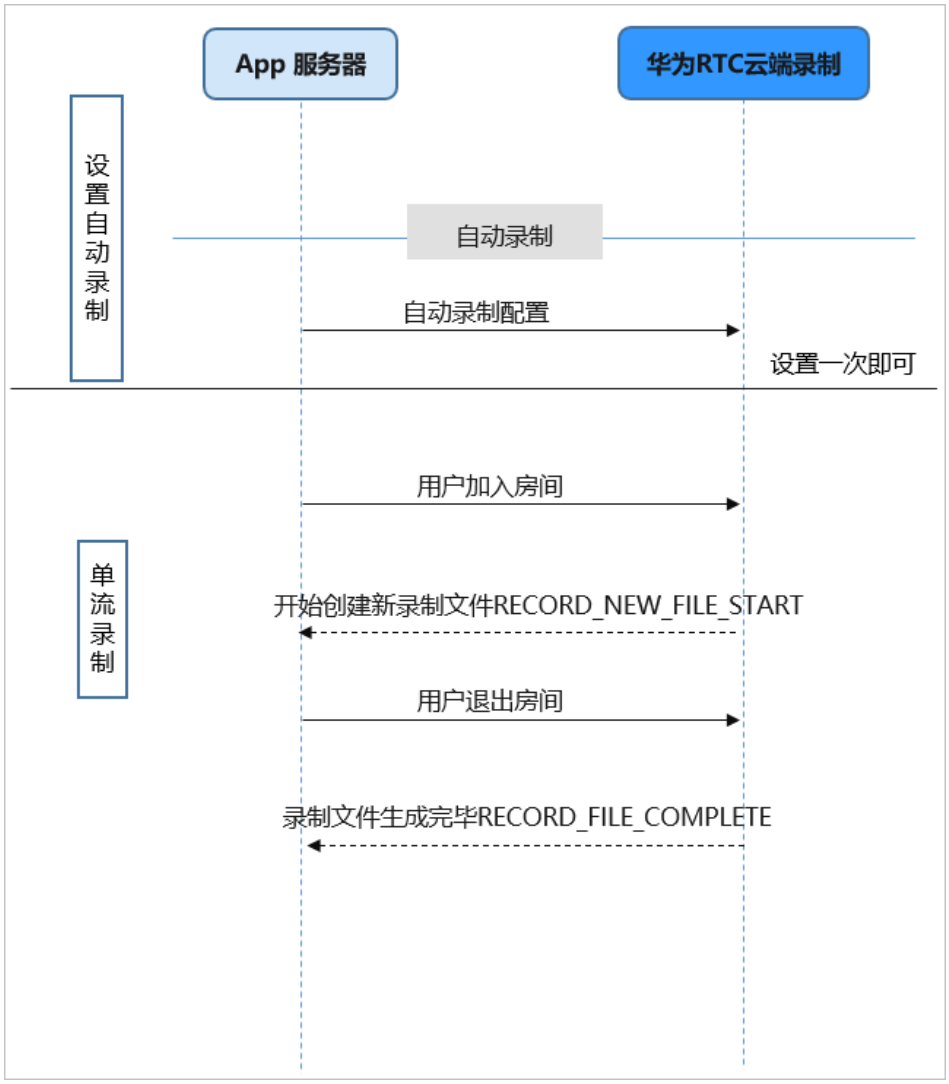
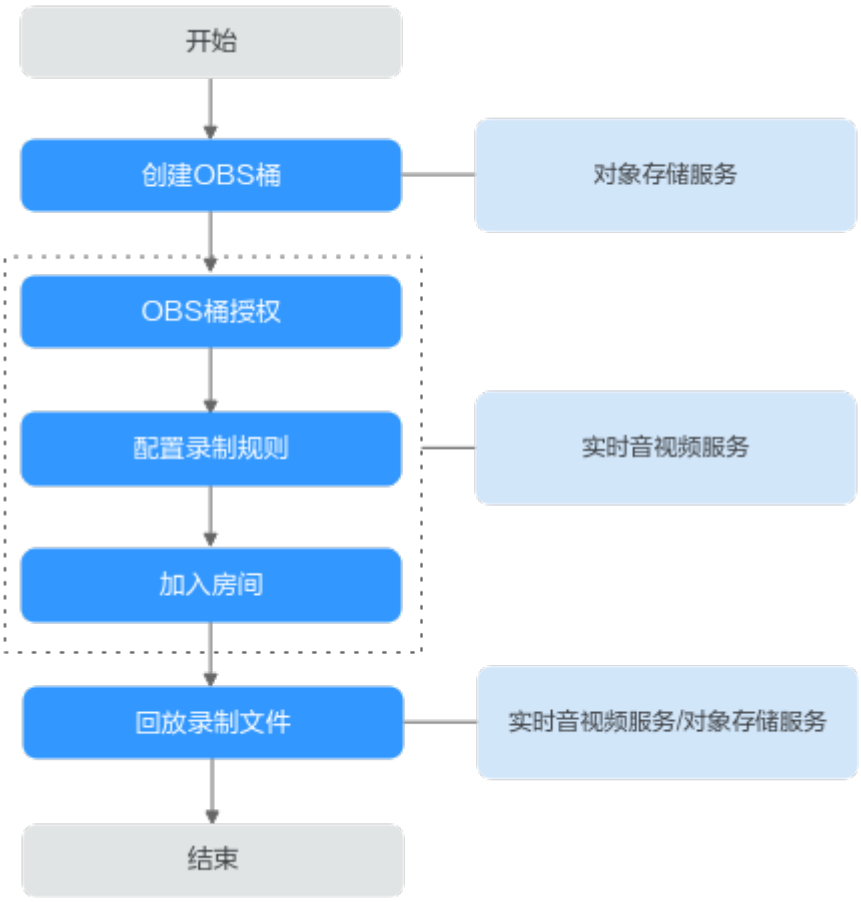


图 1-2 单流自动录制



实现流程



1. **创建OBS桶**：创建用于存储SparkRTC录制文件的OBS桶，若已有OBS桶，请直接执行2。

   说明

   由于单AZ桶的可靠性低于多AZ桶，为避免因OBS服务异常导致录制失败，建议您创建多AZ桶用于录制文件的存储。

2. **OBS桶授权**：在SparkRTC服务中对存储录制文件的OBS桶进行授权，允许SparkRTC服务将录制文件存储在对应的OBS桶中。
3. **配置录制规则**：为实时音视频互动配置录制规则，并开启自动录制功能，加入SparkRTC房间后，与应用中的**录制规则ID**相同的录制模板会自动生效，录制内容按录制设置存储至OBS中。还可以通过设置回调地址获取录制任务状态通知。
4. **加入房间**：录制规则配置完成后，您可以通过SparkRTC APP加入某个SparkRTC房间进行音视频互动，SparkRTC会根据配置的录制规则对正在直播的音视频进行录制。

   说明

   若配置录制规则时，未开启自动录制功能，则加入SparkRTC房间后，您需要调用**实时音视频API**开启、查询和控制云端录制任务。SparkRTC服务才会根据接口中的**录制规则ID**对实时音视频互动画面进行录制。

5. **回放录制文件**：录制完成后，在已配置的回调地址中会收到录制任务的回调消息，您可以获取到录制文件的基本信息，也可以在OBS中管理录制文件，如下载、分享、删除等。

> **说明**
>
> 录制文件的分辨率与推流分辨率相关，按推流原分辨率进行录制。

实现步骤

步骤1 请参见**OBS帮助中心**创建桶。若已有OBS桶，请直接执行**步骤2**。

> **说明**
>
> 创建的OBS桶所在区域必须为**华北-北京四**。

步骤2 OBS桶授权。

1. 登录**实时音视频控制台**。
2. 在左侧导航树中选择“云资源授权”，进入桶授权页面。
3. 在对应的OBS桶行单击“授权”，完成桶授权。

步骤3 配置录制规则。

1. 登录**实时音视频控制台**。
2. 在左侧导航树中选择“应用管理”，进入应用管理页面。
3. 在需要创建录制规则的应用行单击“录制配置”，进入录制配置页面。
4. 在“录制规则”页签，单击“添加”，进入添加录制规则页面。

> **说明**
>
> 一个应用ID仅支持创建一个录制规则。

5. 请您按照实际需求配置录制参数，参数说明如**表1-2**所示。

表 1-2 录制参数说明

参数名		描述
存储-桶		存储录制文件的OBS桶。 目前录制文件仅支持存储到**华北-北京四**的OBS桶中。
区域		OBS桶所在的区域。
存储-路径		存储录制文件的OBS桶路径。
录制格式		录制文件的格式，支持HLS和MP4文件格式。
HLS规则	m3u8命名规则	录制m3u8文件的存储路径和文件的前缀。 默认命名格式： `{app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time}` 上述特殊变量的含义如下： - app_id：应用ID。 - record_format：录制格式。 - stream：流名。 - file_start_time：文件生成时间。

参数名		描述
	录制周期	录制时长支持0-720分钟，最小录制周期为1分钟，最大录制周期为12小时，超过12小时，系统将按照命名规则生成新文件。如果录制周期为0，则整个流录制为一个文件。
	最大断流合并时长	支持如下三种配置： – 断流后生成新文件：是指录制的直播流中断后，会立即生成新的录制文件。 – 断流后不生成新文件：是指录制的直播流中断后，会和之前录制的文件合并为一个文件。最大断流合并时长为30天。 – 其他数值：是指录制的直播流中断时间在设置范围内，则和之前录制的文件合并为一个文件，否则，生成新的录制文件。
MP4规则	mp4命名规则	录制mp4文件的存储路径和文件的前缀。 默认命名格式： {app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time} 上述特殊变量的含义如下： – app_id：应用ID。 – record_format：录制格式。 – stream：流名。 – file_start_time：文件生成时间。
	录制周期	录制时长支持1-180分钟，最小录制周期为1分钟，最大录制周期为3小时，超过3小时，系统将按照命名规则生成新文件。
	最大断流合并时长	支持如下两种配置： – 断流后生成新文件：是指录制的直播流中断后，会立即生成新的录制文件。 – 其他数值：是指录制的直播流中断时间在设置范围内，则和之前录制的文件合并为一个文件，否则，生成新的录制文件。

6. 单击“确定”，在录制规则列表中会增加一条新的录制规则。

图 1-3 录制规则

7. 您可以在录制规则列表中，根据实际需求选择是否开启自动录制功能。自动录制功能开启后，若该应用下有新创建的房间，则会按照已配置的录制规则自动对该房间中的实时音视频互动过程进行录制。

> **说明**
>
> 自动录制功能开启后，仅对同一应用下新创建的房间生效，自动录制功能开启前已创建的房间不生效。

步骤4 加入房间。

录制规则配置完成后，您可以通过SparkRTC APP加入某个SparkRTC房间进行音视频互动，SparkRTC服务会根据配置的录制规则对正在直播的音视频进行录制。

> **说明**
>
> 若配置录制规则时，未开启自动录制功能，则加入SparkRTC房间后，您需要调用**实时音视频API**开启云端录制任务，SparkRTC才会根据API中的**录制规则ID**对实时音视频互动画面进行录制。

步骤5 回放录制文件。

录制完成后，您可以在OBS控制台中或通过回调消息查看录制文件。

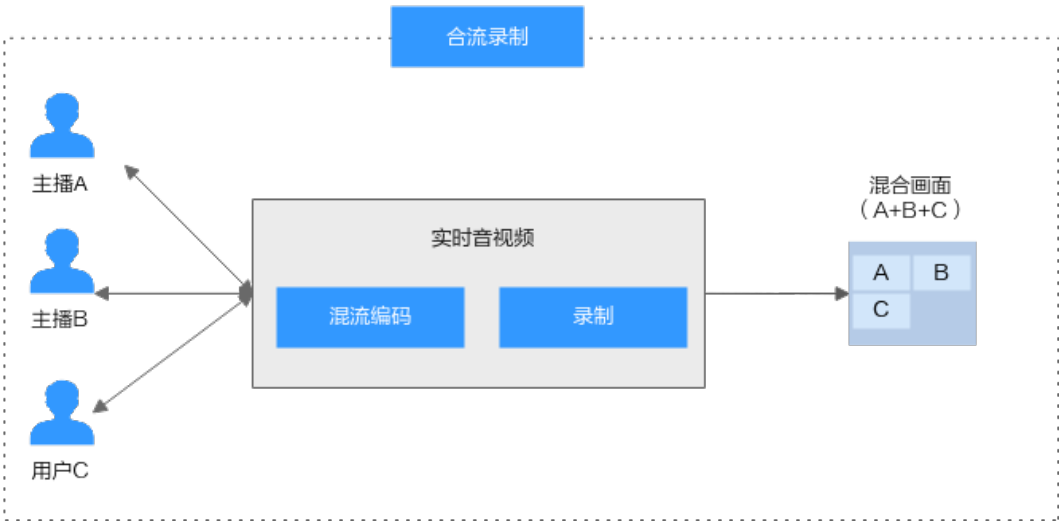
- 通过OBS控制台查看录制文件
  a. 在**OBS管理控制台**左侧导航栏选择“对象存储”。
  b. 在桶列表中单击存储SparkRTC录制文件的桶，进入“概览”页面。
  c. 在左侧导航栏，单击“对象”，查看录制文件信息。
  d. 您还可以对录制文件进行下载、分享等操作，具体请参见**OBS帮助中心**。

----结束

1.3 合流录制

场景说明

将房间中的多路音视频进行云端混流，再将混合后的音视频流录制成一个文件。

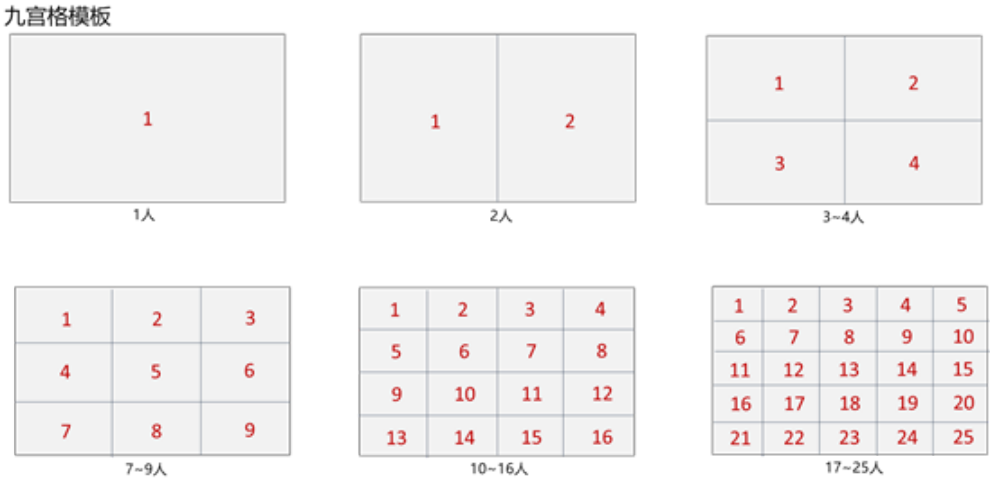


合流布局

在合流录制模式下，支持分屏九宫格模板和共享屏幕模板（主视窗居左/主视窗居右）两种预设合流布局。同时，也支持用户自定义合流布局样式（自定义视频窗格位置）。

- **九宫格模板**

  每个用户画面平铺在画布上，大小一致。根据用户数量，动态调整每个画面的大小和位置。最多支持25个画面。不同人数的实际布局效果如下图所示。

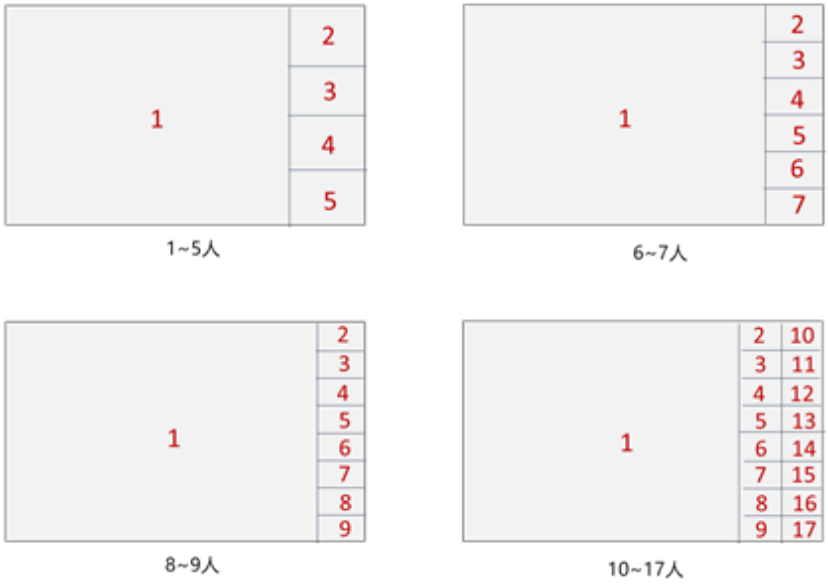


  - 如果实际视频流的宽高比与视窗的宽高比不一致，视频画面会裁剪以适配视窗的大小。
  - 中途有流退出房间，则该条流的画面会被后进入房间的流顶替。
  - 如果房间内的人数不足，则剩余位置显示背景色。
  - 如果用户只发送音频，仍然会占用画面位置。
  - 支持背景图，如果房间内人数不足，显示背景图。

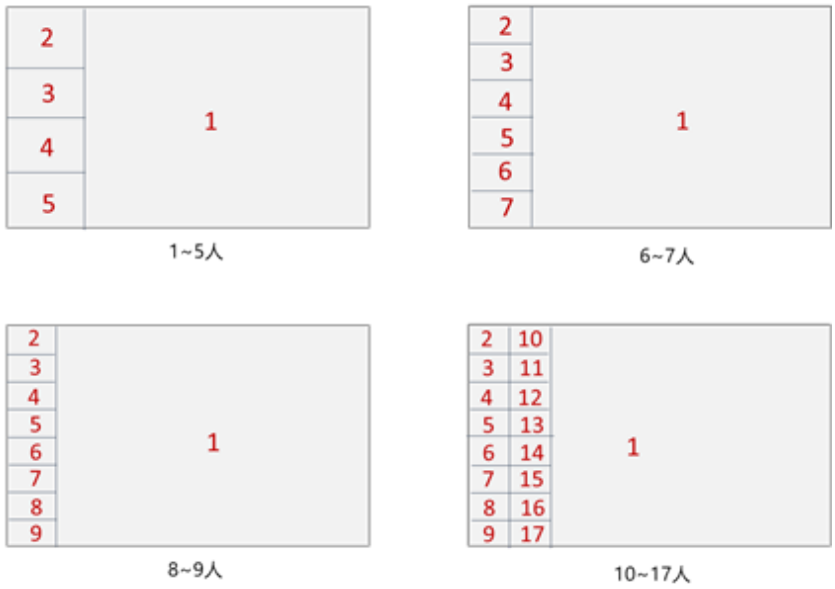
- **屏幕共享模板**

  屏幕分享（或者主讲人摄像头画面）始终占据屏幕左侧或者右侧大画面位置，其他用户依次垂直排列于旁边。最多支持17个画面。不同人数的实际布局效果如下图所示。

屏幕共享模板-大视窗居左——screen_share_left



屏幕共享模板-大视窗右——screen_share_right

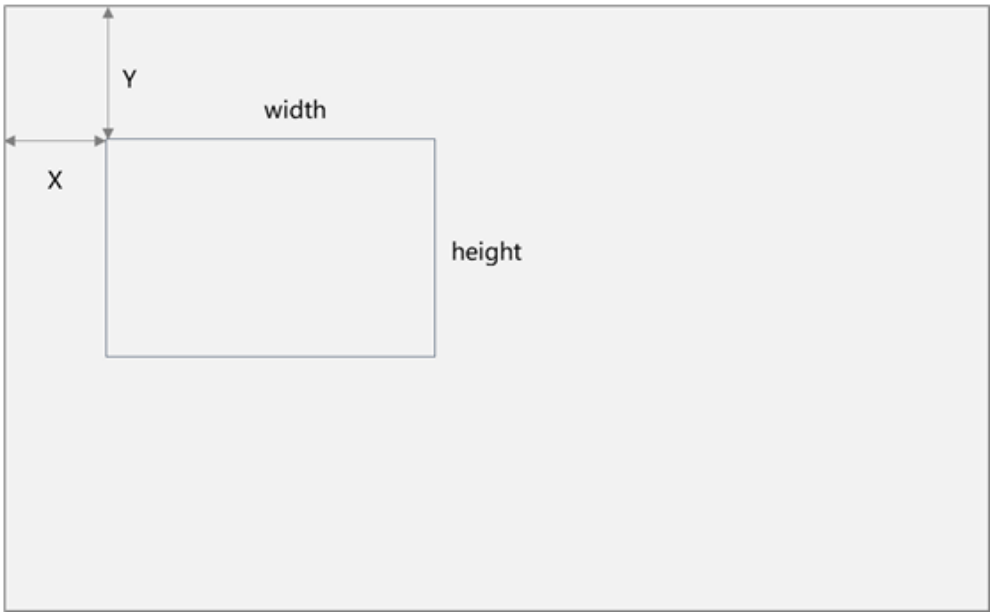


- 大视窗画面，可选择主讲摄像头流或共享屏幕流。
- 大视窗画面，显示指定的UID用户的视频，如果未指定或者指定用户未进入频道，大视窗区域显示背景色。
- 左侧大视窗为了保持内容的完整性采用缩放方式处理，右侧小视窗采用裁剪方式适配视窗的大小。
- 右侧小视窗画面按照加入房间的时间先后顺序排列。
- 右侧小视窗有流退出房间，则该条流的画面会被后进入房间的流顶替。
- 如果房间内的人数不足，则剩余位置显示背景色。
- 如果用户只发送音频，仍然会占用画面位置。

- 支持背景图，如果房间内人数不足，显示背景图。

- **自定义布局模板**

  支持用户自定义合流布局样式，可灵活设置用户画面的大小，指定用户画面在视频画布上的相对位置。

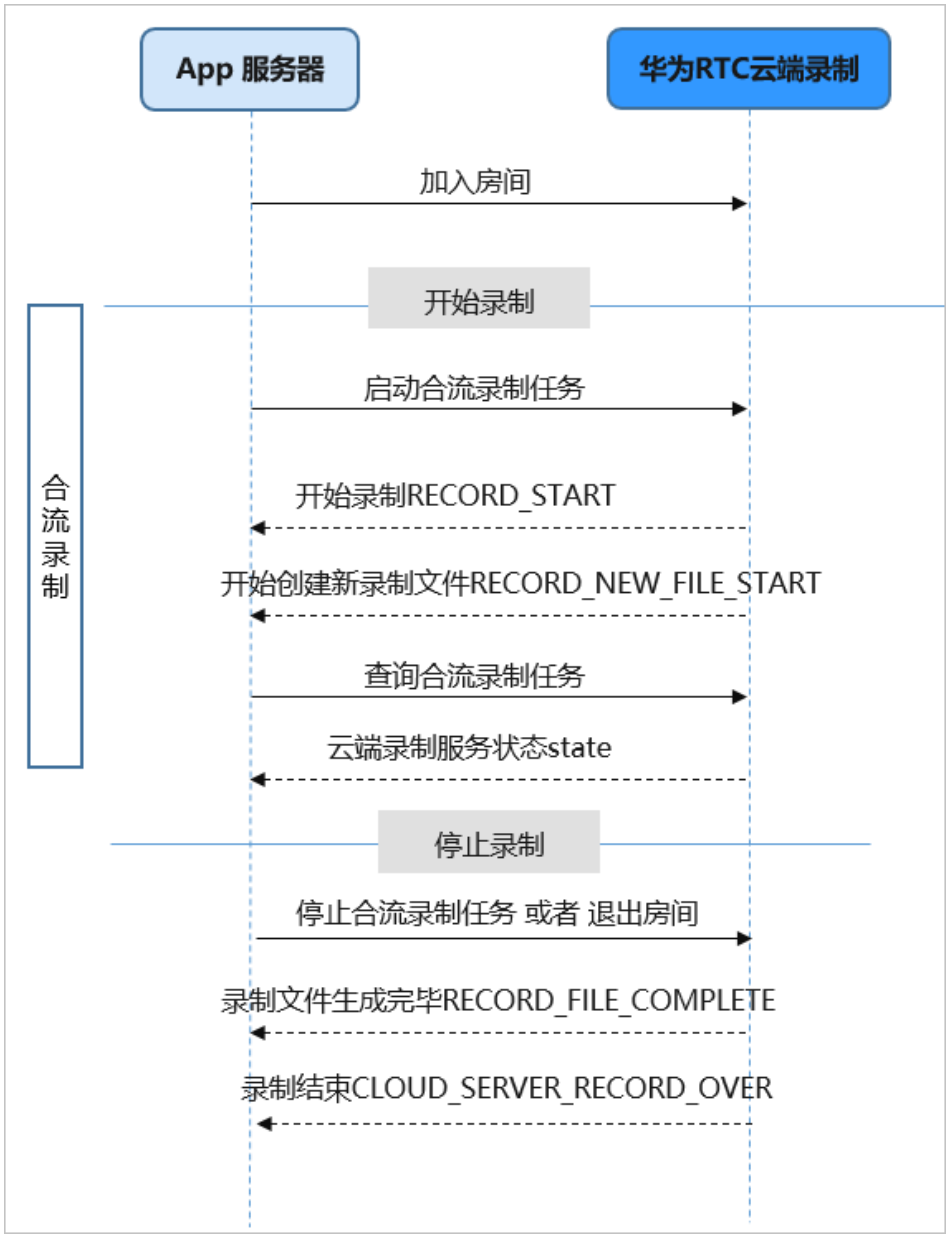
  - 可自定义各个视频窗格在画布上的位置。
  - 可自定义各个视频窗格的宽和高。
  - 针对每一个窗格，可通过user_id指定显示房间内某一用户。
  - 针对窗格，可自定义选择呈现摄像头流或者屏幕分享流。
  - 如果实际视频流的宽高比与视窗的宽高比不一致，自定义布局场景下支持选择裁剪和缩放两种模式。
  - 如果房间内的人数不足，则剩余位置显示背景色。
  - 如果用户只发送音频，仍然会占用画面位置。
  - 支持背景图，如果房间内人数不足，则显示背景图。

录制机制

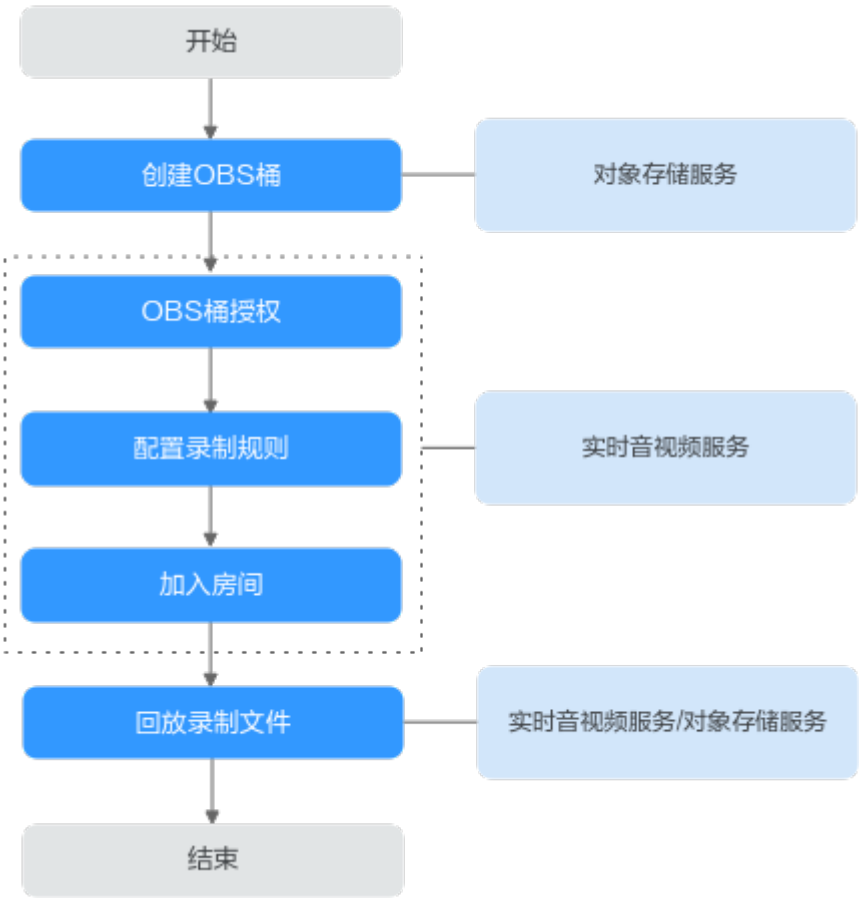
SparkRTC提供的**合流录制**，具体的实现机制如**图1-4**所示。

说明：启动合流录制任务时，需要设定**MixParam**合流参数，指定**layout_template**视频布局模板。

图 1-4 合流录制



实现流程



1. **创建OBS桶**：创建用于存储SparkRTC录制文件的OBS桶，若已有OBS桶，请直接执行2。

   说明

   由于单AZ桶的可靠性低于多AZ桶，为避免因OBS服务异常导致录制失败，建议您创建多AZ桶用于录制文件的存储。

2. **OBS桶授权**：在SparkRTC服务中对存储录制文件的OBS桶进行授权，允许SparkRTC服务将录制文件存储在对应的OBS桶中。
3. **配置录制规则**：为实时音视频互动配置录制规则，并开启自动录制功能，加入SparkRTC房间后，与应用中的**录制规则ID**相同的录制模板会自动生效，录制内容按录制设置存储至OBS中。还可以通过设置回调地址获取录制任务状态通知。
4. **加入房间**：录制规则配置完成后，您可以通过SparkRTC APP加入某个SparkRTC房间进行音视频互动，SparkRTC会根据配置的录制规则对正在直播的音视频进行录制。

   说明

   若配置录制规则时，未开启自动录制功能，则加入SparkRTC房间后，您需要调用**实时音视频API**开启、查询和控制云端录制任务。SparkRTC服务才会根据接口中的**录制规则ID**对实时音视频互动画面进行录制。

5. **回放录制文件**：录制完成后，在已配置的回调地址中会收到录制任务的回调消息，您可以获取到录制文件的基本信息，也可以在OBS中管理录制文件，如下载、分享、删除等。

说明

录制文件的分辨率与推流分辨率相关，按推流原分辨率进行录制。

实现步骤

步骤1 请参见**OBS帮助中心**创建桶。若已有OBS桶，请直接执行**步骤2**。

说明

创建的OBS桶所在区域必须为**华北-北京四**。

步骤2 OBS桶授权。

1. 登录**实时音视频控制台**。
2. 在左侧导航树中选择“云资源授权”，进入桶授权页面。
3. 在对应的OBS桶行单击“授权”，完成桶授权。

步骤3 配置录制规则。

1. 登录**实时音视频控制台**。
2. 在左侧导航树中选择“应用管理”，进入应用管理页面。
3. 在需要创建录制规则的应用行单击“录制配置”，进入录制配置页面。
4. 在“录制规则”页签，单击“添加”，进入添加录制规则页面。

说明

一个应用ID仅支持创建一个录制规则。

5. 请您按照实际需求配置录制参数，参数说明如**表1-3**所示。

表 1-3 录制参数说明

参数名		描述
存储-桶		存储录制文件的OBS桶。 目前录制文件仅支持存储到**华北-北京四**的OBS桶中。
区域		OBS桶所在的区域。
存储-路径		存储录制文件的OBS桶路径。
录制格式		录制文件的格式，支持HLS和MP4文件格式。
HLS规则	m3u8命名规则	录制m3u8文件的存储路径和文件的前缀。 默认命名格式： {app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time} 上述特殊变量的含义如下： - app_id：应用ID。 - record_format：录制格式。 - stream：流名。 - file_start_time：文件生成时间。

参数名		描述
	录制周期	录制时长支持0-720分钟，最小录制周期为1分钟，最大录制周期为12小时，超过12小时，系统将按照命名规则生成新文件。如果录制周期为0，则整个流录制为一个文件。
	最大断流合并时长	支持如下三种配置： – 断流后生成新文件：是指录制的直播流中断后，会立即生成新的录制文件。 – 断流后不生成新文件：是指录制的直播流中断后，会和之前录制的文件合并为一个文件。最大断流合并时长为30天。 – 其他数值：是指录制的直播流中断时间在设置范围内，则和之前录制的文件合并为一个文件，否则，生成新的录制文件。
MP4规则	mp4命名规则	录制mp4文件的存储路径和文件的前缀。 默认命名格式： {app_id}/{record_format}/{stream}_{file_start_time}/{stream}_{file_start_time} 上述特殊变量的含义如下： – app_id：应用ID。 – record_format：录制格式。 – stream：流名。 – file_start_time：文件生成时间。
	录制周期	录制时长支持1-180分钟，最小录制周期为1分钟，最大录制周期为3小时，超过3小时，系统将按照命名规则生成新文件。
	最大断流合并时长	支持如下两种配置： – 断流后生成新文件：是指录制的直播流中断后，会立即生成新的录制文件。 – 其他数值：是指录制的直播流中断时间在设置范围内，则和之前录制的文件合并为一个文件，否则，生成新的录制文件。

6. 单击“确定”，在录制规则列表中会增加一条新的录制规则。

图 1-5 录制规则

7. 您可以在录制规则列表中，根据实际需求选择是否开启自动录制功能。自动录制功能开启后，若该应用下有新创建的房间，则会按照已配置的录制规则自动对该房间中的实时音视频互动过程进行录制。

> **说明**
>
> 自动录制功能开启后，仅对同一应用下新创建的房间生效，自动录制功能开启前已创建的房间不生效。

步骤4 加入房间。

录制规则配置完成后，您可以通过SparkRTC APP加入某个SparkRTC房间进行音视频互动，SparkRTC服务会根据配置的录制规则对正在直播的音视频进行录制。

> **说明**
>
> 若配置录制规则时，未开启自动录制功能，则加入SparkRTC房间后，您需要调用**实时音视频API**开启云端录制任务，SparkRTC才会根据API中的**录制规则ID**对实时音视频互动画面进行录制。

步骤5 回放录制文件。

录制完成后，您可以在OBS控制台中或通过回调消息查看录制文件。

- 通过OBS控制台查看录制文件
  a. 在**OBS管理控制台**左侧导航栏选择“对象存储”。
  b. 在桶列表中单击存储SparkRTC录制文件的桶，进入“概览”页面。
  c. 在左侧导航栏，单击“对象”，查看录制文件信息。
  d. 您还可以对录制文件进行下载、分享等操作，具体请参见**OBS帮助中心**。

----结束

2 实现音视频通话

2.1 实现音视频通话（Android）

2.1.1 环境准备

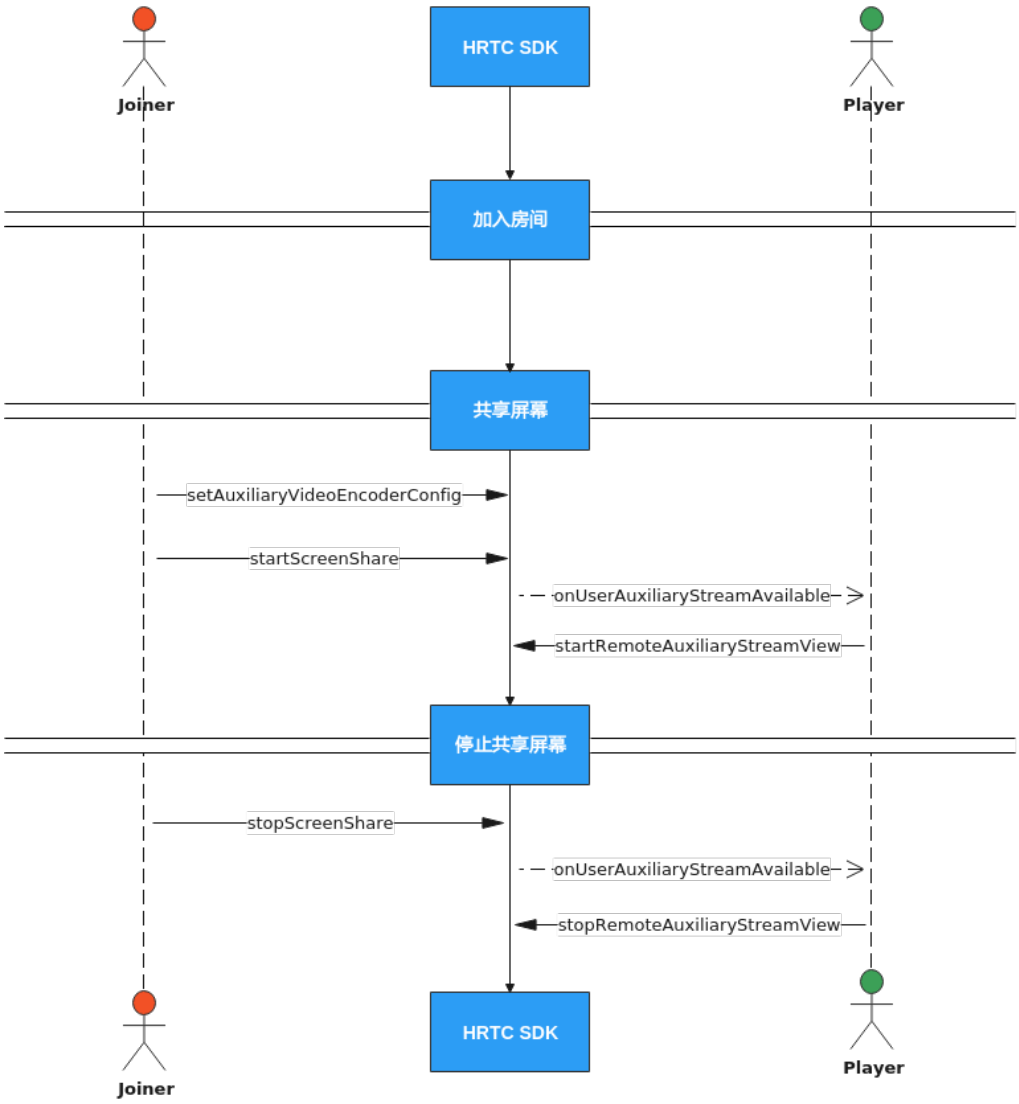
详情请参考**开发前准备**。

2.1.2 屏幕分享

功能描述

屏幕共享用于在音视频会议中，把一个与会者的屏幕内容，以视频的方式分享给其他与会者。

接口调用流程



实现屏幕共享

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **共享桌面**

   加入房间后可以调用setAuxiliaryVideoEncoderConfig设置编码参数，其中HRTCVideoAuxiliarEncParam需要传入宽高、帧率、码率，然后调用startScreenShare开始共享。

```
public void startScreenShare(){
        // 设置编码参数
    mHwRtcEngine.setAuxiliaryVideoEncoderConfig(new HRTCVideoAuxiliarEncParam(widthInt,
heightInt, framerateInt, bitrateInt));
    // 开始共享
    mHwRtcEngine.startScreenShare();
}
```

3. **共享程序**

   暂不支持

4. **接收远端用户的共享流**

   收到远端用户开启共享流回调onUserAuxiliaryStreamAvailable后，可以调用startRemoteAuxiliaryStreamView来设置远端用户的共享流的窗口句柄并开始选看。

   还可以调用updateRemoteAuxiliaryStreamRenderMode设置窗口显示共享流的方式。

```
@Override
public void onUserAuxiliaryStreamAvailable(String roomId, String userId, boolean available){
    if (available) {
        // 设置远端用户视图
    mHwRtcEngine.startRemoteAuxiliaryStreamView(userId, surface);
    // 设置远端用户视图显示模式
        mHwRtcEngine.updateRemoteAuxiliaryStreamRenderMode(userId,
HRTCEnums.HRTCVideoDisplayMode.HRTC_VIDEO_DISPLAY_MODE_HIDDEN,
HRTCEnums.HRTCVideoMirrorType.HRTC_VIDEO_MIRROR_TYPE_DISABLE);
    }
}
```

5. **停止屏幕共享**

   屏幕共享结束时，可以调用stopScreenShare停止屏幕共享。

```
public void stopScreenShare() {
    // 停止屏幕共享
    mHwRtcEngine.stopScreenShare();
}
```

6. **停止接收远端用户的屏幕共享流**

   收到onUserAuxiliaryStreamAvailable消息后，如果选看的远端屏幕共享流不可用，或者收到远端用户下线通知onUserOffline，则接收端必须调用stopRemoteAuxiliaryStreamView关闭共享流窗口视图。

   如果接收端想主动停止接收远端用户的共享流，也需要调用stopRemoteAuxiliaryStreamView。

```
@Override
public void onUserAuxiliaryStreamAvailable(String roomId, String userId, boolean available){
    if (!available) {
        // 关闭共享流窗口视图
    mHwRtcEngine.stopRemoteAuxiliaryStreamView(userId);
    }
}
```

API 参考

startScreenShare

stopScreenShare

startRemoteAuxiliaryStreamView

stopRemoteAuxiliaryStreamView

2.1.3 通话中质量监测

功能描述

加入频道后，SDK会每隔2秒自动触发通话质量相关的回调，上报当前本地和远端的音视频统计信息。

接口调用流程

实现通话中质量监测

1. **通话质量上报**

onNetworkQualityNotify，房间内客户端网络质量实时上报，默认开启，每2s上报一次，两人以上才会回调。

```
@Override
public void onNetworkQualityNotify(List<HRTCQualityInfo> localQuality, List<HRTCQualityInfo>
remoteQuality) {
    // 将本地和远端质量信息显示到屏幕上
}
```

2. **本地音频统计信息**

   onLocalAudioStatsNotify，本地音频流详情，2s触发一次回调。

```
@Override
public void onLocalAudioStatsNotify(List<HRTCLocalAudioStats> localStats) {
    // 将本地音频统计消息显示到屏幕上
}
```

3. **远端音频统计信息**

   onRemoteAudioStatsNotify，远端音频流详情，2s触发一次回调。

```
@Override
public void onRemoteAudioStatsNotify(List<HRTCRemoteAudioStats> remoteStats) {
    // 将远端音频统计消息显示到屏幕上
}
```

4. **本地视频统计信息**

   onLocalVideoStatsNotify，本地视频流详情，2s触发一次回调。

```
@Override
public void onLocalVideoStatsNotify(List<HRTCLocalVideoStats> localStats) {
    // 将本地视频统计消息显示到屏幕上
}
```

5. **远端视频统计信息**

   onRemoteVideoStatsNotify，远端视频流详情，2s触发一次回调。

```
@Override
public void onRemoteVideoStatsNotify(List<HRTCRemoteVideoStats> remoteStats) {
    // 将远端视频统计消息显示到屏幕上
}
```

6. **本地音频和远端音频统计信息**

   onAudioStatsNotify回调返回的参数中，有本地当前用户的音频信息，也有远端用户的音频信息。

   音频信息包括：码率、丢包、延迟、抖动。

```
@Override
public void onAudioStatsNotify(List<HRTCLocalAudioStats> localStats, List<HRTCRemoteAudioStats>
remoteStats){
    // 将本地和远端音频统计信息刷新显示到屏幕上
}
```

7. **本地视频和远端视频统计信息**

   onVideoStatsNotify回调返回的参数中，有本地当前用户的视频信息，也有远端用户的视频信息。

   视频信息包括：码率、分辨率、帧率、丢包、延迟、抖动。

```
@Override
public void onVideoStatsNotify(List<HRTCLocalVideoStats> localStats, List<HRTCRemoteVideoStats>
remoteStats){
    // 将本地和远端视频统计信息刷新显示到屏幕上
}
```

API 参考

onNetworkQualityNotify

onLocalAudioStatsNotify

onRemoteAudioStatsNotify

onLocalVideoStatsNotify

onRemoteVideoStatsNotify

onAudioStatsNotify

onVideoStatsNotify

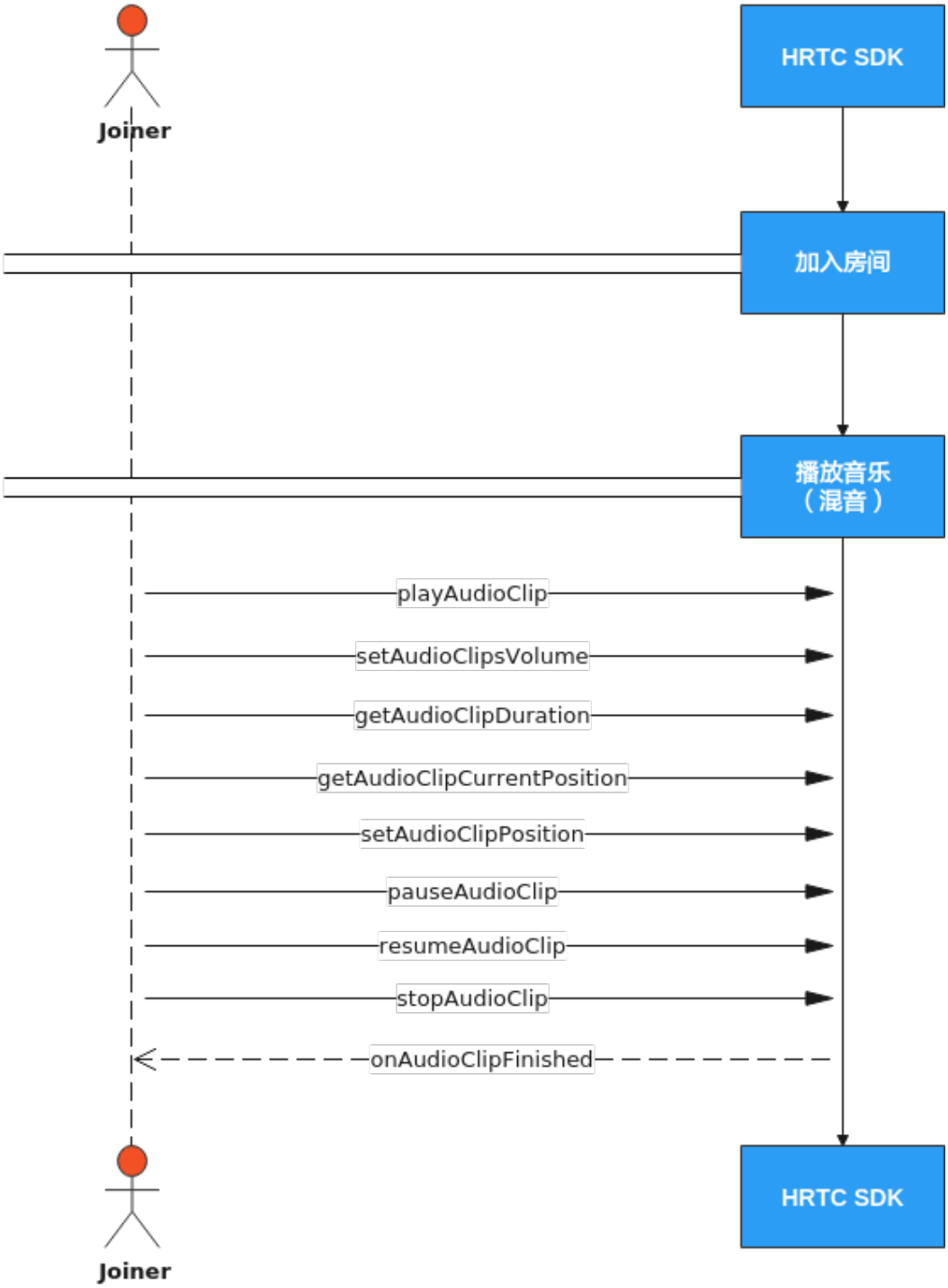
2.1.4 播放音效文件

功能描述

用户可以同时播放多个音效文件，给自己和其他与会者听，用于烘托气氛。

支持本地或在线文件路径，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程



实现播放音效文件

1. **加入房间**
   参考**接口调用流程**中加入房间的时序图步骤加入房间。
2. **播放音效文件**
   调用playAudioClip播放音效文件并启动混音。
   可以同时播放多个音效文件，不同音效文件用不同的soundId参数进行区别。

```
public void playClip() {
    int soundId = 0; // 音效文件Id
    mHwRtcEngine.playAudioClip(soundId, filePath, loop, 0, 0, 100, publish, 0);
}
```

3. **设置音效文件音量**

   调用setVolumeOfAudioClip设置音效播放的音量。

   其中，progress取值范围为0 ~ 100。

```
public void setClipVolume() {
    mHwRtcEngine.setVolumeOfAudioClip(soundId, progress);
}
```

4. **获取音效文件总时长**

   播放过程中，调用getAudioClipDuration获取音效文件总时长，可用于刷新界面上的进度条。

```
public void refreshSeekBar() {
    int duration = mHwRtcEngine.getAudioClipDuration(filePath);
    // 刷新进度条
}
```

5. **获取音效文件播放位置**

   播放过程中，调用getAudioClipCurrentPosition获取音效文件当前播放位置，可用于刷新界面上的进度条。

```
public void refreshSeekBar() {
    int pos = mHwRtcEngine.getAudioClipCurrentPosition(soundId);
    // 刷新进度条
}
```

6. **设置音效文件播放位置**

   播放过程中，调用setAudioClipPosition设置音效文件播放位置，可用于跳转至对应播放位置。

```
public void seekTo() {
    mHwRtcEngine.setAudioClipPosition(soundId, effectPosition);
}
```

7. **暂停播放音效文件**

   播放过程中，调用pauseAudioClip暂停播放指定的音效文件，调用pauseAllAudioClips暂停所有的音效文件。

```
public void pauseClip() {
    mHwRtcEngine.pauseAudioClip(soundId);
}

public void pauseAllClip() {
    mHwRtcEngine.pauseAllAudioClips();
}
```

8. **恢复播放暂停的音效文件**

   暂停时，调用resumeAudioClip恢复播放指定的音效文件，调用resumeAllAudioClips恢复所有的音效文件。

```
public void resumeClip() {
    mHwRtcEngine.resumeAudioClip(soundId);
}

public void resumeAllClip() {
    mHwRtcEngine.resumeAllAudioClips();
}
```

9. **停止播放音效文件**

   播放过程中，调用stopAudioClip停止播放指定的音效文件，调用stopAllAudioClips停止所有的音效文件。

```
public void stopClip() {
    mHwRtcEngine.stopAudioClip(soundId);
```



```
}

public void stopAllClip() {
    mHwRtcEngine.stopAllAudioClips();
}
```

10. **音效文件播放结束回调**

    在音效播放结束后，会收到onAudioClipFinished回调

```
@Override
public void onAudioClipFinished(int soundId) {
    // 音效播放结束
}
```

API 参考

playAudioClip

stopAudioClip

pauseAudioClip

resumeAudioClip

stopAllAudioClips

pauseAllAudioClips

resumeAllAudioClips

setVolumeOfAudioClip

setAudioClipPosition

getAudioClipCurrentPosition

getAudioClipDuration

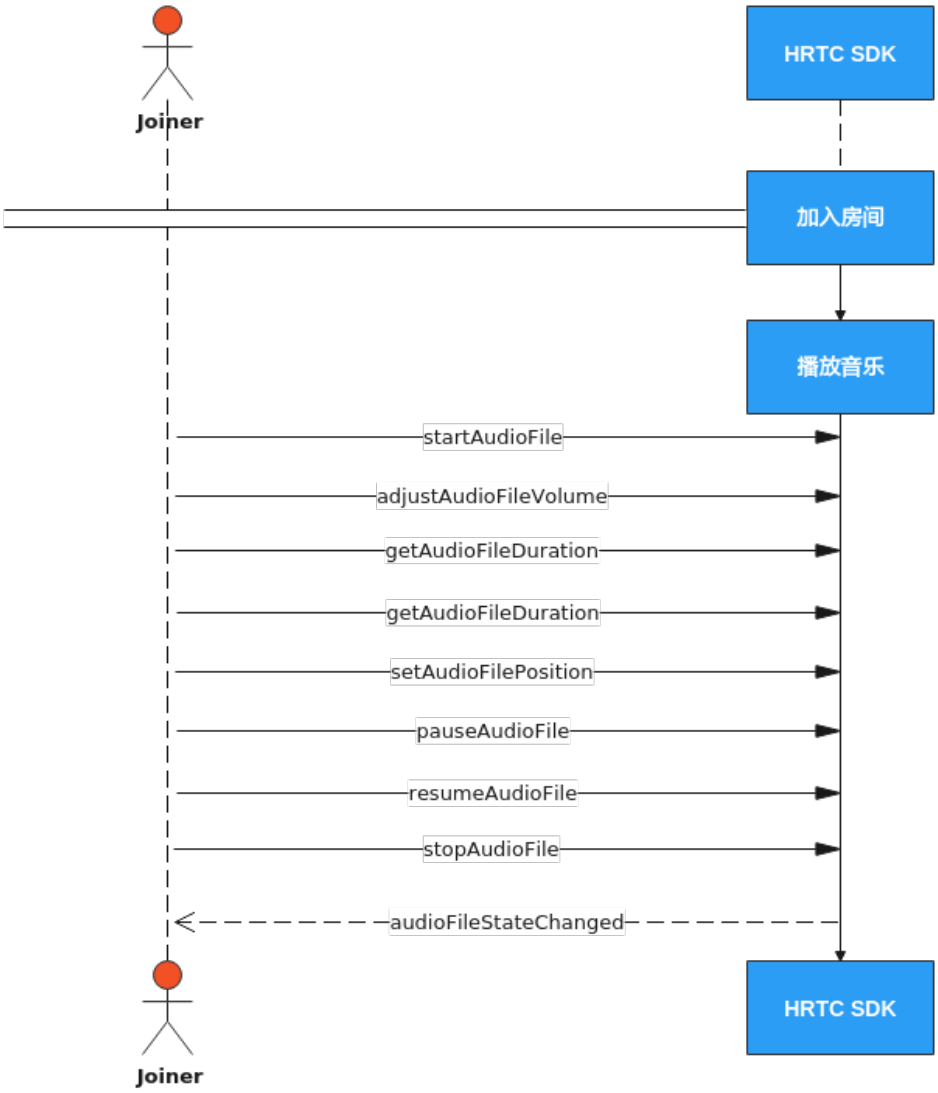
2.1.5 播放音乐文件

功能描述

混音是将音乐文件和麦克风音频混合，一般用于播放时长较长的背景音乐或者伴奏。同一时间只能播放一个音乐文件。可以在本地播放，也可以同时播放给其他与会者听。

可以播放本地或在线音乐文件，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程



实现播放音乐文件

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音乐文件**

   调用startAudioFile播放音乐文件，目前仅支持本端播放。

   返回值为0，则说明调用成功，不为0则表示失败。

```
public void startAudio() {
    int ret = mHwRtcEngine.startAudioFile(audioFilePaths[currentFilePos], publish, cycle, replace);
    if (ret == 0) {
        // 成功
    } else {
        // 失败
    }
}
```

3. **设置音乐文件音量**

   调用adjustAudioFileVolume调整播放音量。

其中，progress取值范围为0 ~ 100。

```
public void setAudioVolume() {
    mHwRtcEngine.adjustAudioFileVolume(progress);
}
```

4. **获取音乐文件总时长**

   播放过程中，调用getAudioFileDuration获取音乐文件总时长，可用于刷新界面上的进度条。

```
public void refreshSeekBar() {
    int duration = mHwRtcEngine.getAudioFileDuration();
    // 刷新进度条
}
```

5. **获取音乐文件播放位置**

   播放过程中，调用getAudioFilePosition获取音乐文件当前播放位置，可用于刷新界面上的进度条。

```
public void refreshSeekBar() {
    int pos = mHwRtcEngine.getAudioFilePosition();
    // 刷新进度条
}
```

6. **设置音乐文件播放位置**

   播放过程中，调用setAudioFilePosition设置音乐文件播放位置，可用于跳转至对应播放位置。

```
public void seekTo() {
    mHwRtcEngine.setAudioFilePosition(pos);
}
```

7. **暂停播放音乐文件**

   播放过程中，调用pauseAudioFile暂停播放当前音乐文件。

```
public void pauseAudio() {
    mHwRtcEngine.pauseAudioFile();
}
```

8. **恢复播放暂停的音乐文件**

   暂停时，调用resumeAudioFile恢复播放当前音乐文件。

```
public void resumeAudio() {
    mHwRtcEngine.resumeAudioFile();
}
```

9. **停止播放音乐文件**

   播放过程中，调用stopAudioFile停止播放当前音乐文件。

```
public void stopAudio() {
    mHwRtcEngine.stopAudioFile();
}
```

10. **音乐文件播放结束回调**

    播放结束后，SDK会触发onAudioMixStateChangedNotify回调来通知上层应用。

```
@Override
public void onAudioMixStateChangedNotify(HRTCEnums.HRTCAudioFileState state,
HRTCEnums.HRTCAudioFileReason reason, long value) {
    // 判断state，如果是HRTC_AUDIO_FILE_STOPPED，则表示收到音乐文件播放结束通知，可以刷新界面，比如恢复播放前初始界面状态。
}
```

API 参考

startAudioFile

stopAudioFile

pauseAudioFile

resumeAudioFile

adjustAudioFileVolume

getAudioFileDuration

getAudioFilePosition

setAudioFilePosition

onAudioMixStateChangedNotify

2.1.6 原始音频数据（音频前后处理）

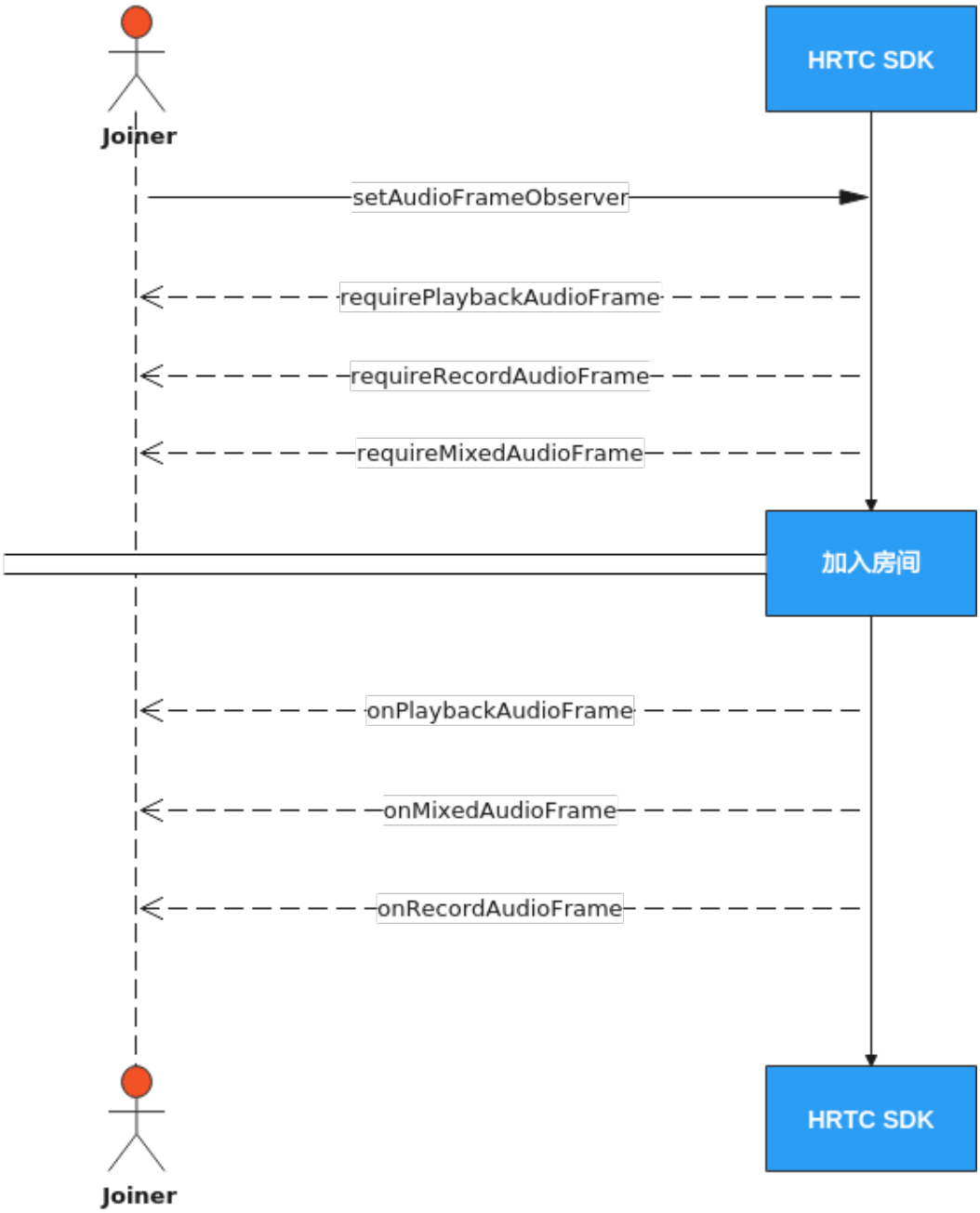
功能描述

音视频传输过程中，可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行音频前处理，然后发送给远端。也可以进行音频后处理。

接口调用流程



实现原始音频数据（音频前后处理）

1. **注册音频前后处理**

   获取mHwRtcEngine的MediaEngine对象，调用setAudioFrameObserver方法进行注册。

   传入的参数是需要实现了IHRTCAudioFrameObserver的实例对象。

   从回调中获取音频帧，并进行处理。

```
@Override
public void onAudioFramePlayback(HRTCAudioFrame hrtcAudioFrame) {
    // 需要播放的音频数据回调，从接口回调中取到音频数据以作后处理
}
```

```
@Override
public void onAudioFrameMixed(HRTCAudioFrame hrtcAudioFrame) {
    // 全部音频混音数据回调，包含上下行所有通道
}

@Override
public void onAudioFrameRecord(HRTCAudioFrame hrtcAudioFrame) {
    // 音频采集原始数据回调，对音频数据的修改会发送到远端
}

@Override
public boolean requirePlaybackAudioFrame() {
    // 是否开启音频后处理
    return false;
}

@Override
public boolean requireRecordAudioFrame() {
    // 是否开启音频前处理
    return false;
}

@Override
public boolean requireMixedAudioFrame() {
    // 是否需要开启全部音频混音数据回调
    return false;
}
```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setAudioFrameObserver

onAudioFramePlayback

onAudioFrameMixed

onAudioFrameRecord

requireRecordAudioFrame

requirePlaybackAudioFrame

requireMixedAudioFrame

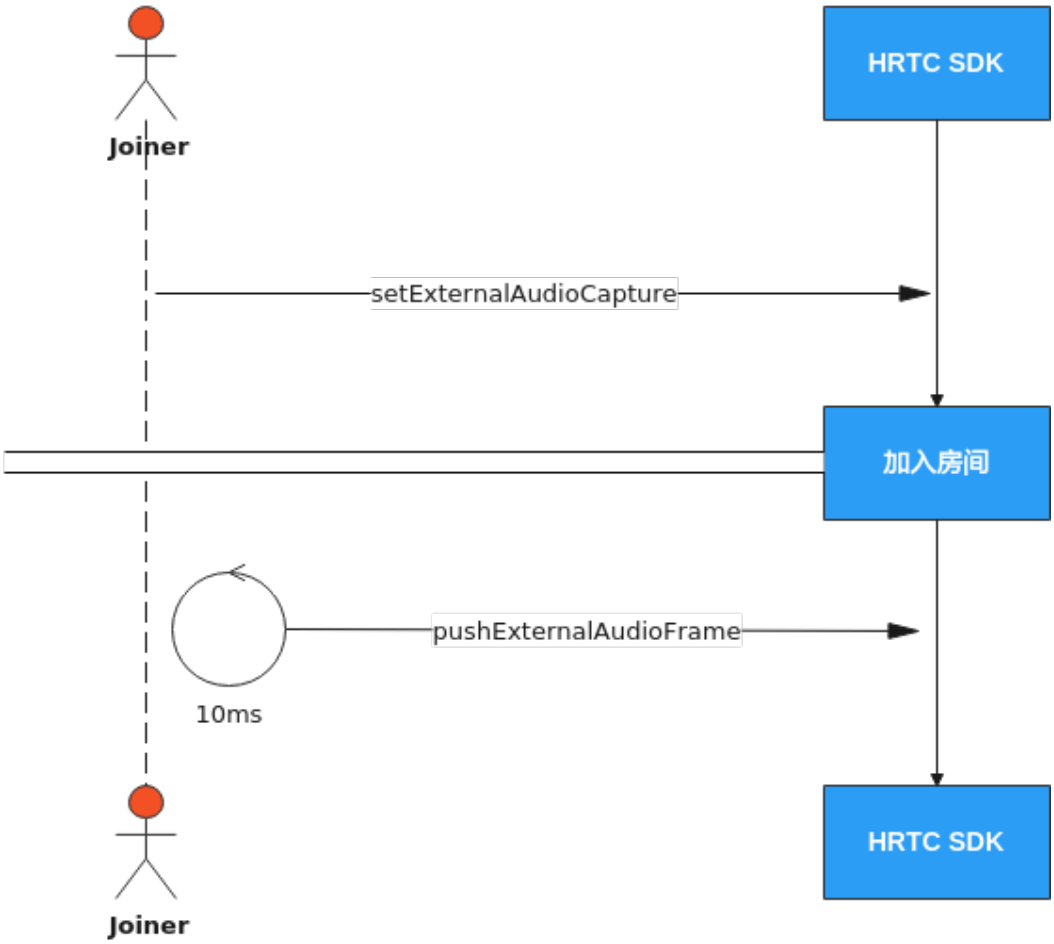
2.1.7 音频自采集和音频自渲染

功能描述

实时音频传输过程中，HWRtcEngine SDK通常会启动默认的音频模块进行采集和渲染。在以下场景中，您可能会发现默认的音频模块无法满足开发需求：

- app中已有自己的音频模块。
- 需要使用自定义的采集或播放处理。
- 某些音频采集设备被系统独占。

接口调用流程



实现音频自采集和音频自渲染

1. **加入房间前**

   加入房间前，调用setExternalAudioCapture，开启音频自采集。

   ```
   public void openExternalAudio() {
       mHwRtcEngine.setExternalAudioCapture(true, sample, 1);
   }
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **定时推送音频数据帧**

   加入房间成功回调后，每隔10ms定时调用pushExternalAudioFrame接口推送外部音频数据。

   音频数据大小：10 * sample * channel * 16 / 8 / 1000字节，其中sample和channel是调用setExternalAudioCapture传入的采样率和声道数。

   ```
   public void pushExternalAudio() {
       new Timer().schedule(new TimerTask() {
               @Override
               public void run() {
                   mHwRtcEngine.pushExternalAudioFrame(audioData);
               }
           }, 0, 10);
   }
   ```

4. **音频自渲染**
   暂不支持

API 参考

setExternalAudioCapture

pushExternalAudioFrame

2.1.8 原始视频数据（视频前后处理）

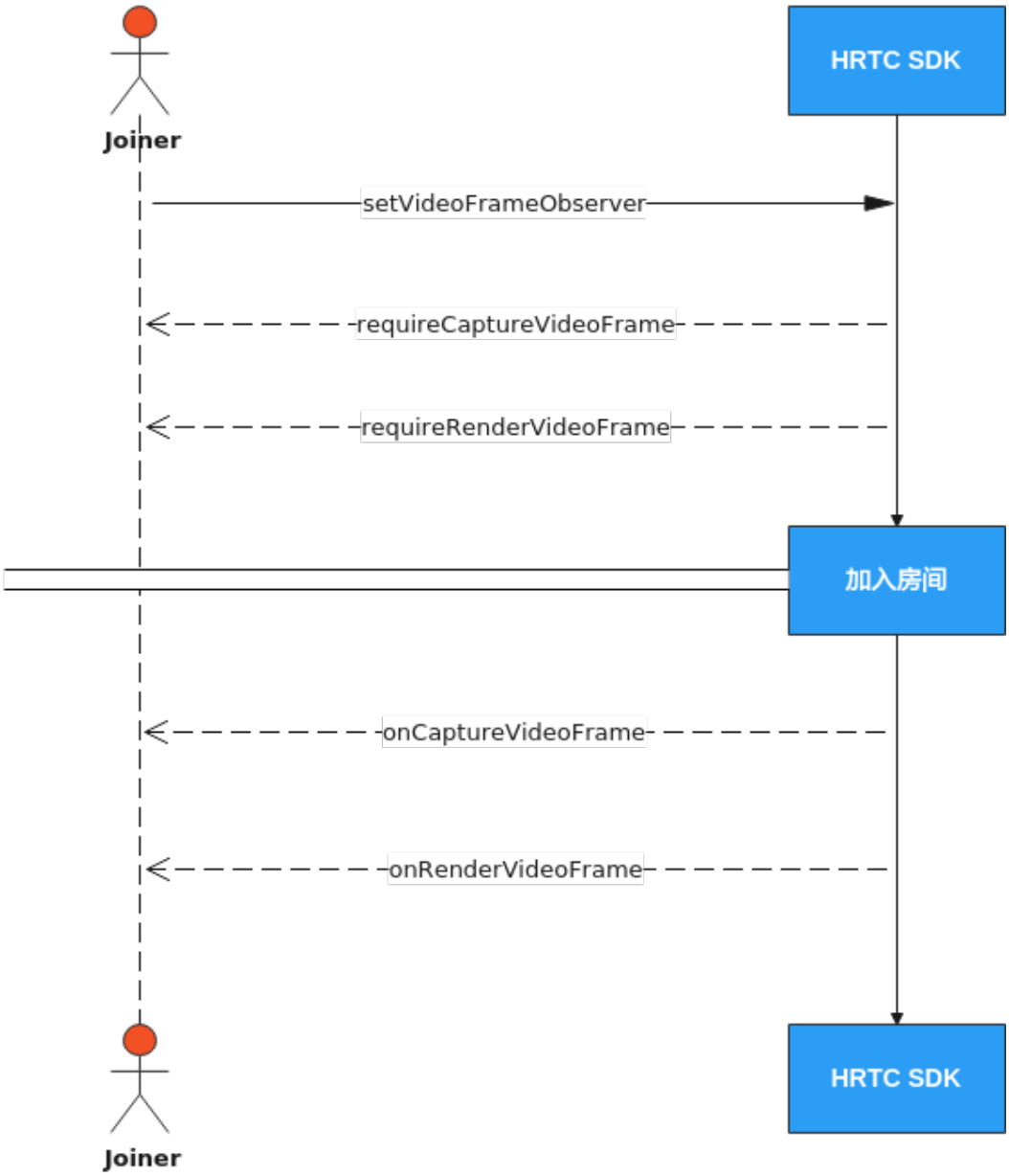
功能描述

音视频传输过程中，可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始视频数据可以进行视频前处理，然后发送给远端。也可以进行视频后处理。

接口调用流程



实现原始视频数据（视频前后处理）

1. **注册视频前后处理**

   获取mHwRtcEngine的MediaEngine对象，调用setVideoFrameObserver方法进行注册。

   传入的参数是需要实现了IHRTCVideoFrameObserver的实例对象。

   从回调中获取视频帧，并进行处理。

```
@Override
public void onVideoFrameCapture(HRTCVideoFrame hrtcVideoFrame) {
    // 原始视频回调，从接口回调中取到原始视频数据以作前处理
}

@Override
public void onVideoFrameRender(String s, HRTCVideoFrame hrtcVideoFrame) {
```

```
    // 原始视频数据处理后回调
}

@Override
public boolean requireCaptureVideoFrame() {
    // 是否需要开启前处理
    return false;
}

@Override
public boolean requireRenderVideoFrame() {
    // 是否需要开启后处理
    return false;
}
```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setVideoFrameObserver

onVideoFrameCapture

onVideoFrameRender

requireCaptureVideoFrame

requireRenderVideoFrame

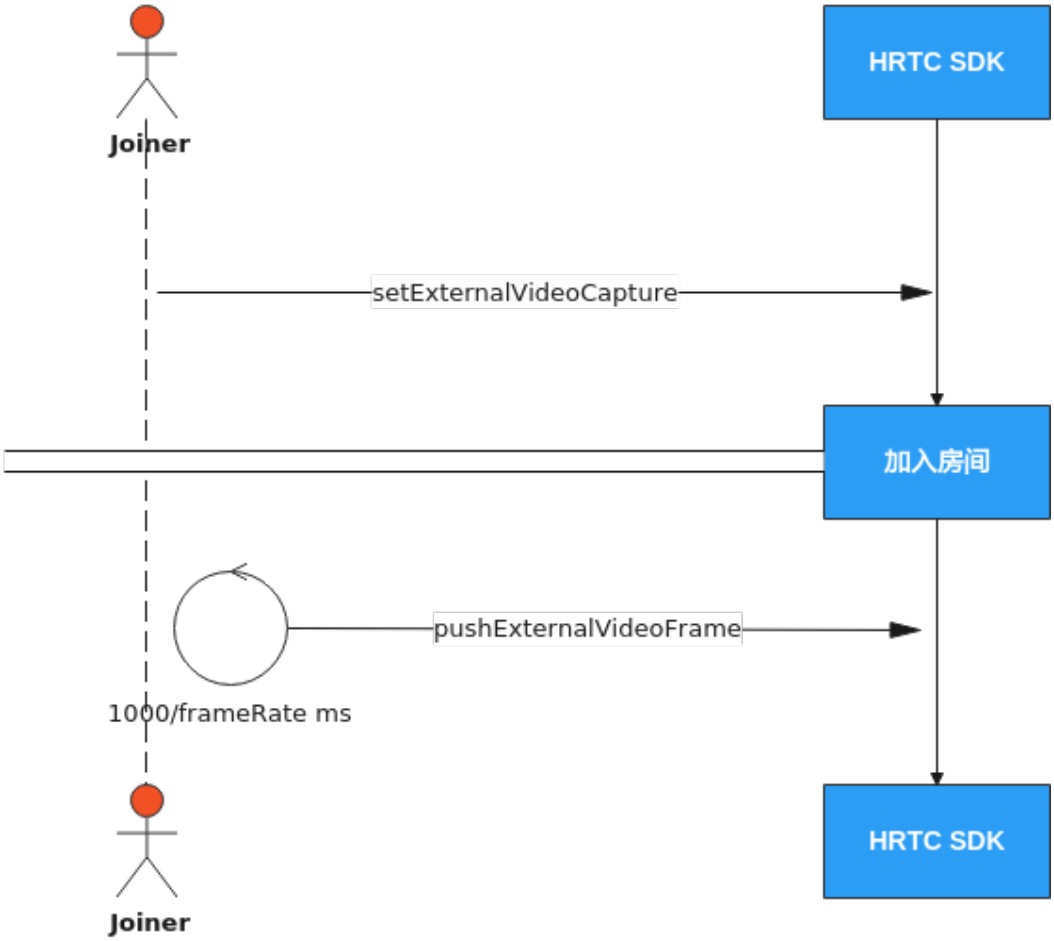
2.1.9 自定义视频采集

功能描述

实时视频传输过程中，HWRtcEngine SDK通常会启动默认的视频模块进行采集和渲染。在以下场景中，您可能会发现默认的视频模块无法满足开发需求：

- app中已有自己的视频模块。
- 需要使用自定义的采集或播放处理。
- 某些视频采集设备被系统独占。

接口调用流程



实现自定义视频采集

1. **加入房间前**

   加入房间前，调用setExternalVideoCapture开启视频自采集。

```
public void openExternalVideo() {
    mHwRtcEngine.setExternalVideoCapture(true);
}
```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **定时推送视频数据帧**

   加入房间成功回调后，每隔1000/frameRate ms定时调用pushExternalVideoFrame接口推送外部视频数据。

   其中，frameRate表示帧率。

```
public void pushExternalVideo() {
    new Timer().schedule(new TimerTask() {
            @Override
            public void run() {
                mHwRtcEngine.pushExternalVideoFrame(videoData);
            }
        }, 0, 1000 / frameRate);
}
```

API 参考

setExternalVideoCapture

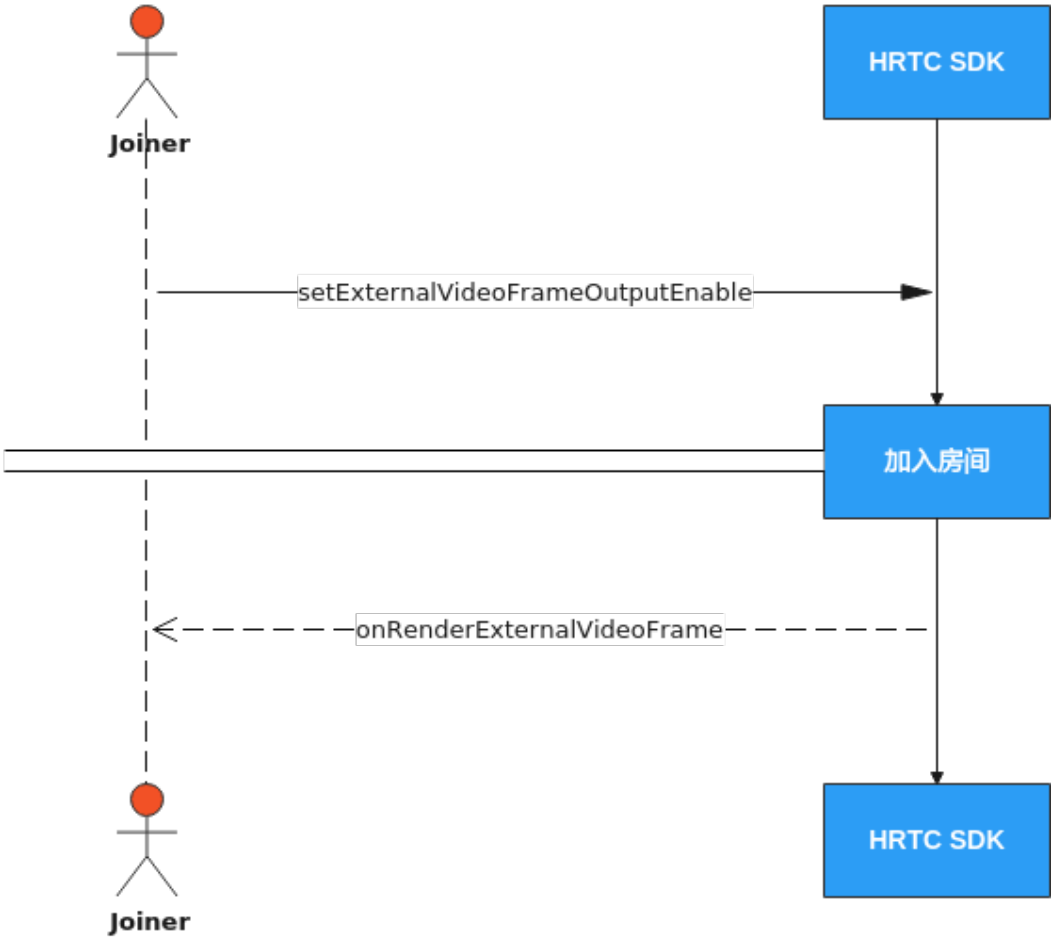
pushExternalVideoFrame

2.1.10 自定义视频渲染

功能描述

实时音视频传输过程中，上层应用可以不用SDK默认的渲染功能，选择对视频帧数据进行自定义渲染。

接口调用流程



实现自定义视频渲染

1. **加入房间前**

   加入房间前，调用setExternalVideoFrameOutputEnable开启视频自渲染。

```
public void openExternalVideoOutput() {
    mHwRtcEngine.setExternalVideoFrameOutputEnable(true, true,
            new HRTCImageBufferFormat(frameFormat, HRTC_VIDEO_IMAGE_BUFFER_BYTE_ARRAY));
}
```

2. **加入房间**

参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **渲染远端视频流**

   加入房间后，可以通过在onRenderExternalVideoFrame回调中进行视频帧的渲染。

```
public void onRenderExternalVideoFrame(String roomId, HRTCEnums.HRTCMediaDirection direction,
String userId, HRTCVideoFrame videoFrame) {
    // 渲染视频数据
}
```

API 参考

onRenderExternalVideoFrame

2.1.11 加入多频道（跨房）

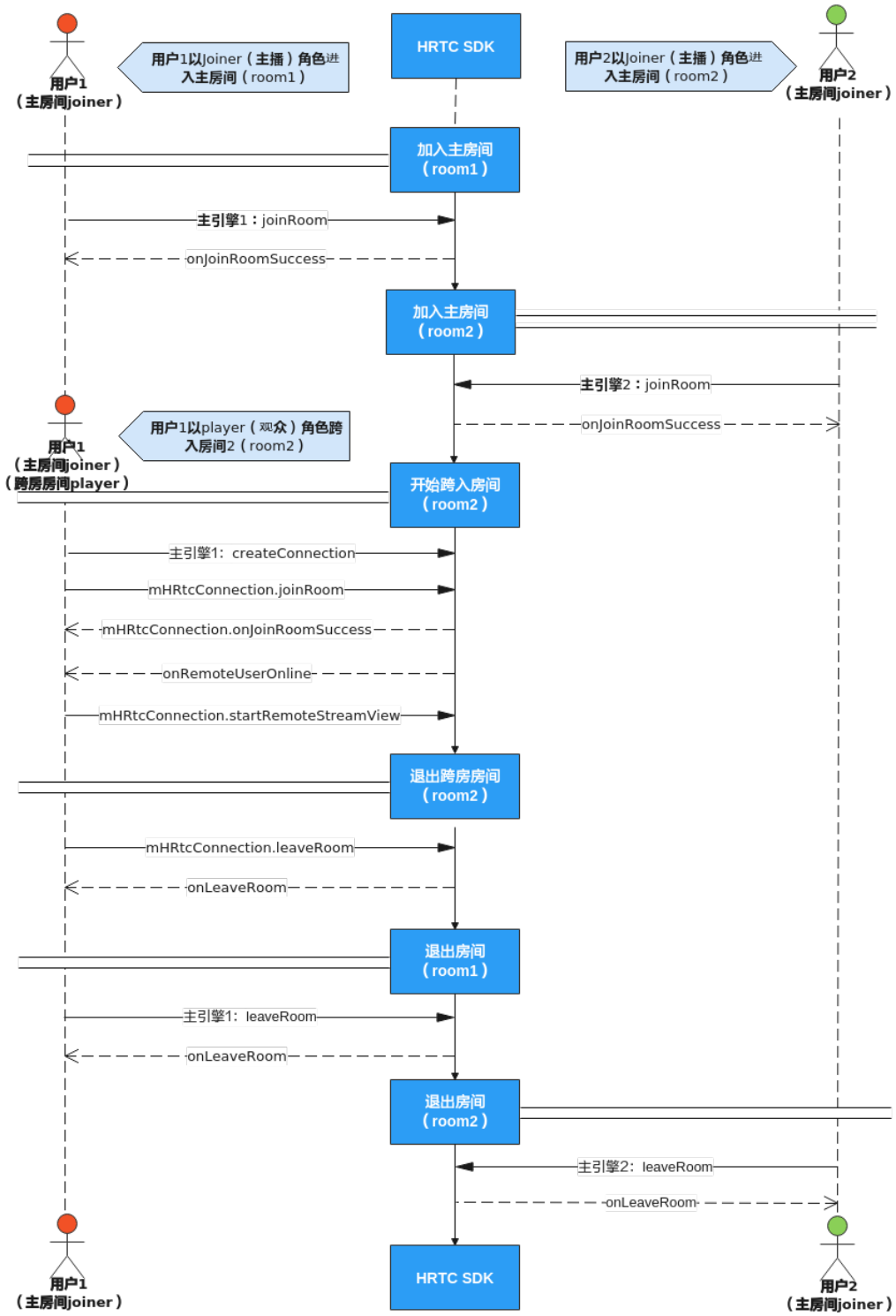
功能描述

跨房间连麦，指主播的媒体流可以同时转发进多个（目前最多支持四个）房间频道，实现主播跨频道与其他主播实时互动的场景。房间中的所有主播可以看见彼此，房间中的观众可以看到所有主播。

注意事项

- 同一时间最多只能创建4个连接对象，每个连接对象对应的房间ID必须互不相同。
- 如果使用connection对象加入房间，则加入房间的房间ID不能和已创建连接对象对应的房间ID相同。
- 同一时间只能以一个JOINER角色加入某一个房间。如果本端在其他房间里的角色是HWRtcRoleJoiner，则需要调用该房间的setUserRole方法将本端在该房间的角色切换为HWRtcRolePlayer后才能以HWRtcRoleJoiner跨入其他房间。如果本端用户是以HWRtcRolePlayer角色加入此跨房房间，则需要调用跨房连接的setUserRole方法将本端的角色切换为HWRtcRoleJoiner后才能发送音频流和视频流。

接口调用流程



实现加入多频道（跨房）

1. **加入主房间**

   参考**接口调用流程**中加入房间的时序图步骤加入主房间。

2. **创建跨房连接**

加入主房间后调用createConnection创建跨房对象，并进行相关参数配置。

其中，HRTCEncryptionConfig需要进行一些参数配置，主要包括：cryptionMode、cryptionSec、suiteType

```
public void setting() {
    mHwRtcConnection = mHwRtcEngine.createConnection(roomid, ((RtcApplication)
getApplication()).getConnectionHandler());
    mHwRtcConnection.setEncryption(new HRTCEncryptionConfig());
}
```

3. **调用跨房连接**

   创建跨房对象后，调用joinRoom接口加入房间。

   其中，HRTCJoinParam需要进行一些参数配置，主要包括：userId、username、roomId

   ```
   public void joinRoom() {
       mHwRtcConnection.joinRoom(new HRTCJoinParam());
   }
   ```

   加入房间成功后，会收到跨房成功回调onJoinRoomSuccess

   加入房间失败后，会收到跨房失败回调onJoinRoomFailure

4. **接收远端用户的视频流**

   当收到远端用户加入房间后，会触发onRemoteUserOnline回调，在回调内，对远端用户调用startRemoteStreamView设置远端窗口并开启收流。

   ```
   @Override
   public void onRemoteUserOnline(HRTCConnection conn, String userId, String userName) {
       mHwRtcConnection.startRemoteStreamView(userId, surface, type, !isRemoteAdaptive);
   }
   ```

5. **退出跨房房间**

   跨房结束后，调用跨房对象的leaveRoom离开房间。

   ```
   public void leaveRoom() {
       mHwRtcConnection.leaveRoom();
   }
   ```

API 参考

createConnection

joinRoom

onJoinRoomSuccess

onJoinRoomFailure

onRemoteUserOnline

startRemoteStreamView

leaveRoom

2.2 实现音视频通话（iOS）

2.2.1 环境准备

详情请参考**开发前准备**。

2.2.2 屏幕共享

功能描述

基于苹果的Replaykit方案，支持用户分享整个系统的屏幕内容，但需要App额外提供一个Extension扩展组件，可实现跨应用屏幕共享。

创建 App Group

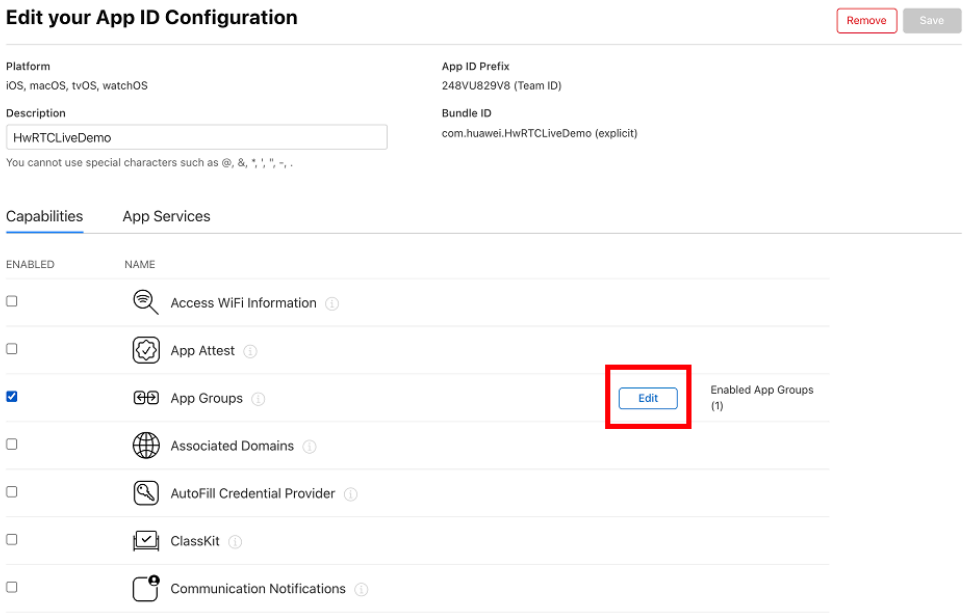
由于华为云SparkRTC是通过App Goup进行进程间的数据通信的，所以需要创建一个App Group。

使用您的APP ID账号登录**https://developer.apple.com/** ，进行证书配置App Group。

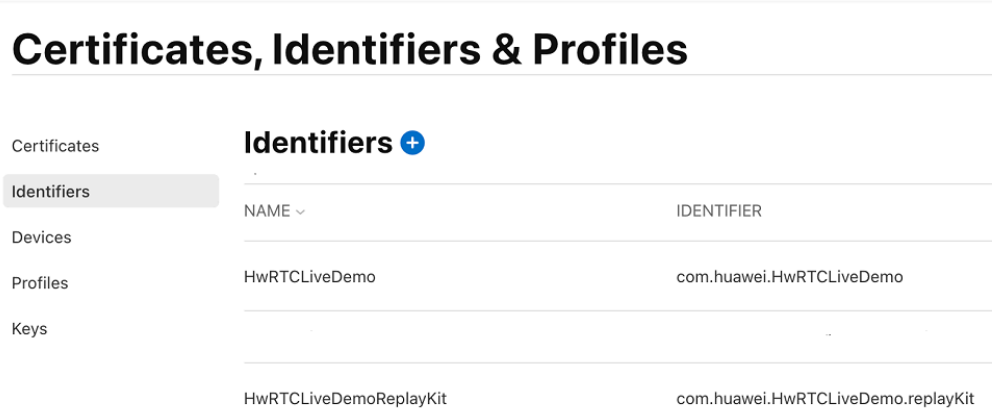
注意：完成后需要重新下载对应的Provisioning Profile。

1. 单击“Certificates, Identifiers & Profiles”，进入“Certificates，Identifiers &Profiles”界面。
2. 在右侧界面中选择“Identifiers”，然后单击 。

3. 选择“App Groups > Continue”。
4. 在弹出的表单中填写“Description”和“Identifier”，其中，Identifier需要传入接口中对应的App Group参数。
5. 填写完成后，单击“Continue”。
6. 返回“Identifiers”页面，在右上角的菜单中选择“App IDs”，然后单击您的App ID（主App与Extension的AppID需要进行同样的配置）。
7. 选中“App Groups”并单击“Edit”。



8. 在弹出的表单中选择您之前创建的App Group，单击“Continue”返回编辑页，再单击“Save”保存。



9. 重新下载Provisioning Profile并配置到XCode中。

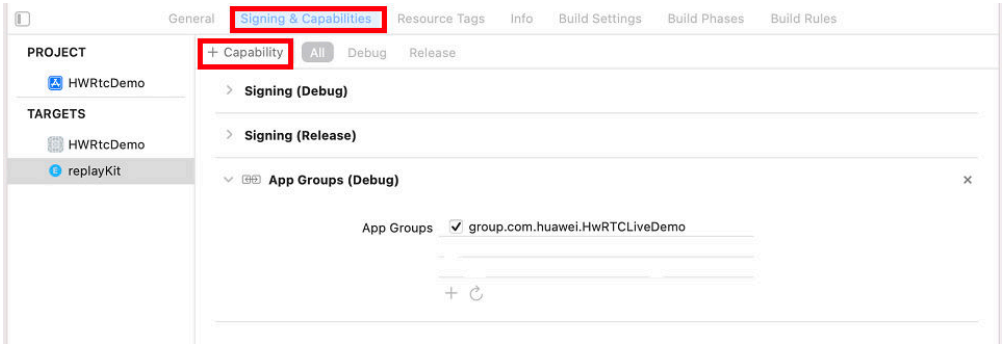
   注意：Extension证书也需要支持App Group。

创建 Broadcast Upload Extension

1. 在Xcode菜单中选择“File > New > Target... > Broadcast Upload Extension”。
2. 在弹出的对话框中填写相关信息，不用勾选“Include UI Extension”，然后单击“Finish”完成创建。

   bundle identifier规则需要遵循主App的bundle identigfier后面拼接上**.xxxx**。
3. 将下载的SDK包中的**HWRtcEngineReplayKit.framework**拖动到工程中，勾选刚刚新创建的Target。
4. 选中新增加的Target，单击“+ Capability”，双击“App Groups”，并选择您创建的AppGroup。

   如下图所示，操作完成后，会在文件列表中生成一个名为*Target名.entitlements*的文件。



5. 选中主App的Target，并按照上述步骤对主App的Target做同样的处理。
6. 在新创建的Target中，Xcode会自动创建一个名为“SampleHandler.h” 的文件，请您用如下代码进行替换。

   注意：需将代码中的APPGROUP改为上文中的创建的App Group Identifier。

```
#import "SampleHandler.h"
#import <ReplayKit/ReplayKit.h>
#import <HWRtcEngineReplayKit/HWRtcReplay.h>

#define HWReplayAppGroupId    @"group.com.huawei.HwRTCLiveDemo"
@interface SampleHandler() < HWRtcReplayDelegate >
@property (nonatomic, assign)BOOL connectRet;
@end

@implementation SampleHandler

- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *, NSObject *> *)setupInfo {
    // User has requested to start the broadcast. Setup info from the UI extension can be supplied but optional.
    [[HWRtcReplay sharedInstance] setupWithAppGroup:HWReplayAppGroupId delegate : self];
}

-(void)broadcastPaused {
    // User has requested to pause the broadcast. Samples will stop being delivered.
}

-(void)broadcastResumed {
    // User has requested to resume the broadcast. Samples delivery will resume.
}

-(void)broadcastFinished {
    // User has requested to finish the broadcast.
    [[HWRtcReplay sharedInstance] broadcastFinished];
}

-(void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType :(RPSampleBufferType)sampleBufferType {

    switch (sampleBufferType) {
    case RPSampleBufferTypeVideo:
        // Handle video sample buffer
        [[HWRtcReplay sharedInstance] sendVideoSampleBuffer:sampleBuffer];
        break;
    case RPSampleBufferTypeAudioApp:
        // Handle audio sample buffer for app audio
        break;
    case RPSampleBufferTypeAudioMic:
        // Handle audio sample buffer for mic audio
        break;

    default:
        break;
    }
}
```

```
-(void)replayBroadcastFinished {
    NSError *error;
    [self finishBroadcastWithError : error];
}
```

主 APP 接收逻辑

您可以按照如下步骤启动屏幕共享。

1. 调用startScreenShareWithAppGroup:方法，并传入自定义的AppGroup，SDK会进入等待状态。
2. 等待用户触发屏幕分享 ，开始正式屏幕共享。
   屏幕共享需要您在iOS系统的控制中心，通过长按录屏按钮进行触发。

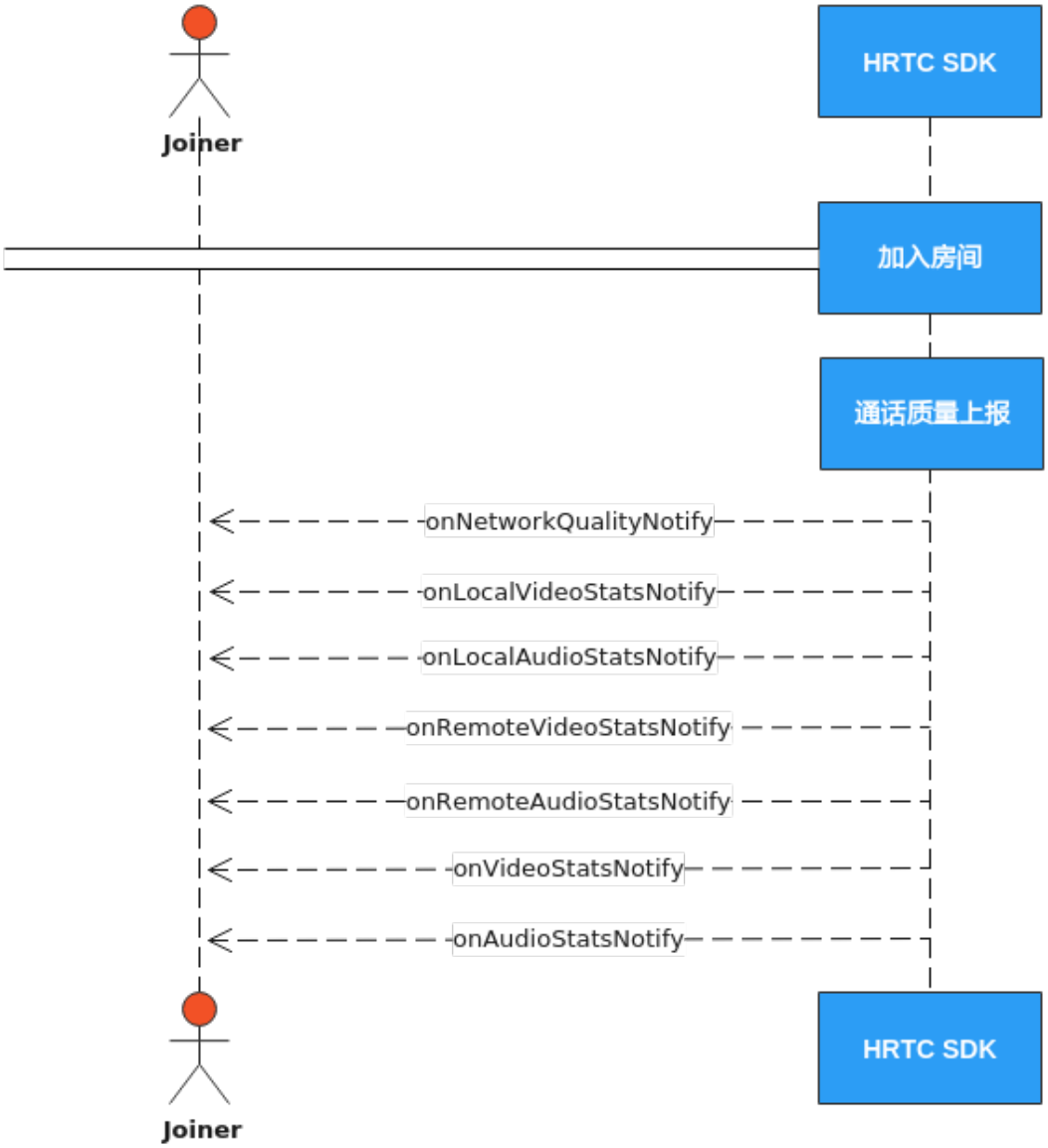
3. 通过调用stopScreenShare接口结束屏幕共享。

2.2.3 通话质量监测

功能描述

加入频道后，SDK会每隔2秒自动触发通话质量相关的回调，上报当前通话的网络质量、本地和远端的音视频统计信息。

上报接口



通话质量上报

onNetworkQualityNotify回调上报当前通话中每个入会者的上下行网络质量。默认开启，每2s上报一次。

```
- (void)onNetworkQualityNotify:(NSArray <HWRtcQualityInfo *> * _Nullable)upStreamQualityArray
      downStreamQualityInfo:(NSArray <HWRtcQualityInfo *> * _Nullable)downStreamQualityArray
{
    dispatch_async(dispatch_get_main_queue(),^{
        for (HWRtcQualityInfo *info in upStreamQualityArray) {
            //本地网络质量信息数据处理
        }
        for (HWRtcQualityInfo *info in downStreamQualityArray) {
            //远端网络质量信息数据处理
        }
    });
}
```

本地音频流统计信息报告

onLocalAudioStatsNotify回调上报本地设备发送音频流的统计信息。您可以了解到当前通话声道数（单声道或双声道）、发送音频的采样率、码率、比特率、丢包率、延时和抖动等。

```
- (void)onLocalAudioStatsNotify:(NSArray <HWRtcLocalAudioStats *> * _Nullable)localAudioStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcLocalAudioStats *audioStatsInfo in localAudioStatsArray) {
            //本地音频信息数据处理
        }
    });
}
```

远端音频流统计信息报告

onRemoteAudioStatsNotify回调上报当前通话中每个远端用户音频流的统计信息。您可以了解到每个远端用户发送的音频流的采样率、声道数、码率、丢包率、延时、抖动和卡顿时长等一些信息。

```
- (void)onRemoteAudioStatsNotify:(NSArray <HWRtcRemoteAudioStats *> *
_Nullable)remoteAudioStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcRemoteAudioStats *audioStatsInfo in remoteAudioStatsArray) {
            //远端音频信息数据处理
        }
    });
}
```

本地视频流统计信息上报

onLocalVideoStatsNotify回调向您报告当前本地视频流统计信息，包括帧率、码率、延时、抖动等信息。

```
- (void)onLocalVideoStatsNotify:(NSArray <HWRtcLocalVideoStats *> * _Nullable)localVideoStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcLocalVideoStats *videoStatsInfo in localVideoStatsArray) {
            //本地视频信息数据处理
        }
    });
}
```

远端视频流状态监控

onRemoteVideoStatsNotify回调向您报告当前远端视频流统计信息，包括帧率、码率、延时、抖动和卡顿时长等信息。

```
-(void)onRemoteVideoStatsNotify:(NSArray <HWRtcRemoteVideoStats *> *
_Nullable)remoteVideoStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcRemoteVideoStats *videoStatsInfo in remoteVideoStatsArray) {
            //处理远端视频信息
        });
    }
}
```

视频流状态监控

onVideoStatsNotify回调上报视频流的状态，包括本地上行视频流和远端用户的下行视频流状态。

```
- (void)onVideoStatsNotify:(NSArray<HWRtcVideoStatsInfo *> *)videoStatsArray remoteVideoInfo:
(NSArray<HWRtcVideoStatsInfo *> *)remoteVideoStatsInfos
```

```
{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcVideoStatsInfo *videoStatsInfo in videoStatsArray) {
            //本地视频信息数据处理
        }
        for (HWRtcVideoStatsInfo *remoteVideoStatsInfo in videoStatsArray) {
            //远端视频信息数据处理
        }
    });
}
```

音频流状态监控

onAudioStatsNotify回调上报音频流的状态，包括本地上行音频流和远端用户的下行音频流状态。

```
- (void)onAudioStatsNotify:(NSArray<HWRtcAudioStatsInfo *> *)audioStatsArray remoteAudioInfo:
(NSArray<HWRtcAudioStatsInfo *> *)remoteAudioStatsInfos
{https://support.huaweicloud.com/csdk-rtc/rtc_05_0167.html
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcAudioStatsInfo *audioStatsInfo in audioStatsArray) {
            //本地音频信息数据处理
        }
        for (HWRtcAudioStatsInfo *remoteAudioStatsInfo in remoteAudioStatsInfos) {
            //远端音频信息数据处理
        }
    });
}
```

API 参考

onNetworkQualityNotify

onLocalVideoStatsNotify

onLocalAudioStatsNotify

onRemoteVideoStatsNotify

onRemoteAudioStatsNotify

onVideoStatsNotify

onAudioStatsNotify

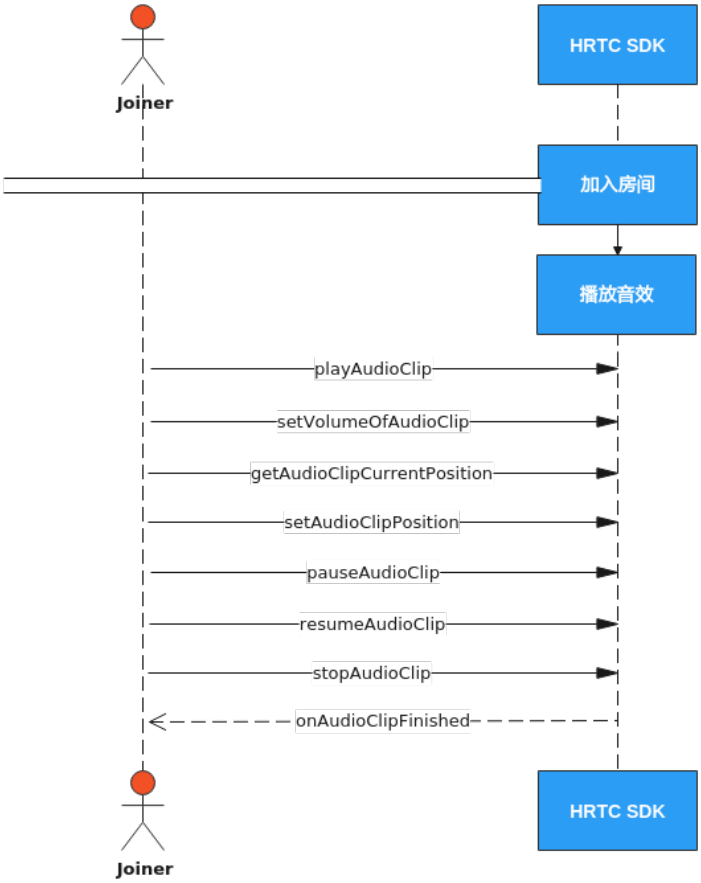
2.2.4 播放音效

功能描述

用户可以同时播放多个音效文件，给自己和其他与会者听，用于烘托气氛。

支持本地或在线文件路径，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程



实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音效文件**

   调用playAudioClip接口可以播放一个音效文件。可以同时播放多个音效文件，不同音效文件用不同的soundId参数进行区别。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine playAudioClip:soundId filePath: [[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]
    loop:1 pitch:0.00 pan:0.00 gain:100 publish:1 startPos:0];
   ```

3. **设置音效文件音量**

   音效文件播放过程中，可以调用setVolumeOfAudioClip设置音效文件播放音量。

   音量大小，范围为0-100。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine setVolumeOfAudioClip:soundId volume:50];
   ```

4. **获取音效文件总时长**

   播放过程中可以调用getAudioClipDuration获取音效文件总时长，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int duration = [rtcEngine getAudioClipDuration: [[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]];
   //根据当前总时长刷新界面进度条
   ```

5. **获取音效文件播放位置**

   播放过程中可以调用getAudioClipCurrentPosition获取音效文件播放位置，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int pos = [rtcEngine getAudioClipCurrentPosition:soundId];
   //根据当前播放位置刷新界面进度条
   ```

6. **设置音效文件播放位置**

   播放过程中可以调用setAudioClipPosition设置音效文件播放位置，可用于通过拖动进度条改变音效文件的播放位置。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine setAudioClipPosition:soundId pos:50];
   ```

7. **暂停播放音效文件**

   调用pauseAudioClip接口可以暂停播放一个音效文件。调用pauseAllAudioClips暂停播放所有正在播放的音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine pauseAudioClip:soundId];
   ```

8. **恢复播放暂停的音效文件**

   音频文件暂停播放后，可以调用resumeAudioClip接口可以恢复播放之前暂停的音效文件。或者调用resumeAllAudioClips恢复播放所有暂停的音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine resumeAudioClip:soundId];
   ```

9. **停止播放音效文件**

   调用stopAudioClip接口停止播放一个音效文件，或者调用stopAllAudioClips接口停止播放所有音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine stopAudioClip:soundId];
   ```

10. **音效文件播放结束回调**

    音效文件播放结束后，sdk会触发onAudioClipFinished回调通知上层应用。

    ```
    - (void)onAudioClipFinished:(NSInteger)soundId
    {
      //收到音效文件播放结束通知后，可以刷新界面，比如恢复播放前初始界面状态。
    }
    ```

API 参考

playAudioClip

setVolumeOfAudioClip

getAudioClipDuration

getAudioClipCurrentPosition

setAudioClipPosition

pauseAudioClip

pauseAllAudioClips

resumeAudioClip

resumeAllAudioClips

stopAudioClip

stopAllAudioClips

onAudioClipFinished

2.2.5 播放音乐

功能描述

混音是将音乐文件和麦克风音频混合，一般用于播放时长较长的背景音乐或者伴奏。同一时间只能播放一个音乐文件。可以在本地播放，也可以同时播放给其他与会者听。

支持播放本地或在线音乐文件，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程

实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音乐文件**

   调用startAudioFile接口可以播放一个音乐文件。同一时刻只能播放一个音乐文件。

```
[rtcEngine startAudioFile:[[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]
publish:1 cycle:1 replace:0];
```

3. **设置音乐文件音量**

   音乐文件播放过程中，可以调用adjustAudioFileVolume设置音乐文件播放音量。

```
[rtcEngine adjustAudioFileVolume:50];
```

4. **获取音乐文件总时长和播放位置**

   音乐文件打开成功后，sdk会触发一次onAudioMixStateChangedNotify回调，传入state参数为HWRtcAudioFileOpenCompleted，此时可以调用getAudioFileDuration获取音乐文件总时长，可用以刷新界面进度条的总时长。

   音乐文件在播放过程中，sdk每秒会触发一次onAudioMixStateChangedNotify回调，其中的state参数为HWRtcAudioFilePositionUpdate，value参数就是当前的音乐文件播放进度，以毫秒为单位，可以在此回调中刷新界面的播放进度条。

   播放过程中也可以调用getAudioFileCurrentPosition获取音乐文件当前播放位置，可用于刷新界面的播放进度条。

```
- (void)onAudioMixStateChangedNotify:(HWRtcAudioFileState)state reason:
(HWRtcAudioFileReason)reason value:(NSUInteger)value{
    switch (reason) {
        case HWRtcAudioFileReasonNone:
        {
            switch (state) {
                case HWRtcAudioFileOpenCompleted:
                {
                    //获取总时长，并刷新进度条
                    int mixDuration = [self.rtcEngine getAudioFileDuration];
                }
                    break;
                case HWRtcAudioFilePositionUpdate:
                {
                    //使用value参数值刷新界面的播放进度条
                }
                    break;
                case HWRtcAudioFilePlaying:
                {
                    //刷新界面
                }
                    break;
                case HWRtcAudioFilePaused:
                {
                    //刷新界面
                }
                    break;
                case HWRtcAudioFileStopped:
                {
                    //刷新界面
                }
                    break;
                default:
                    break;
            }
        }
            break;
        default:
        {
            //错误信息提示
        }
            break;
    }
}
```

5. **设置音乐文件播放位置**

   播放过程中可以调用setAudioFilePosition设置音乐文件播放位置，可用于通过拖动进度条改变音乐文件的播放位置。

```
[rtcEngine setAudioFilePosition:50];
```

6. **暂停播放音乐文件**

   调用pauseAudioFile接口可以暂停播放一个音乐文件。

   ```
   [rtcEngine pauseAudioFile];
   ```

7. **恢复播放暂停的音乐文件**

   音乐文件暂停播放后，可以调用resumeAudioFile接口恢复播放。

   ```
   [rtcEngine resumeAudioFile];
   ```

8. **停止播放音乐文件**

   调用stopAudioFile接口可以停止播放音乐文件。

   ```
   [rtcEngine stopAudioFile];
   ```

9. **音乐文件播放结束回调**

   音乐文件播放结束后，sdk会触发onAudioMixStateChangedNotify回调来通知上层应用。

   ```
   - (void)onAudioMixStateChangedNotify:(HWRtcAudioFileState)state reason:
   (HWRtcAudioFileReason)reason value:(NSUInteger)value{
       //收到音乐文件播放结束的通知，可以刷新界面，比如恢复到播放前初始界面状态
       if ( state == HWRtcAudioFilePlayCompleted ) {
       }
   }
   ```

API 参考

startAudioFile

adjustAudioFileVolume

getAudioFileDuration

getAudioFileCurrentPosition

setAudioFilePosition

pauseAudioFile

resumeAudioFile

stopAudioFile

onAudioMixStateChangedNotify

2.2.6 原始音频数据（音频前后处理）

功能描述

音视频传输过程中，可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

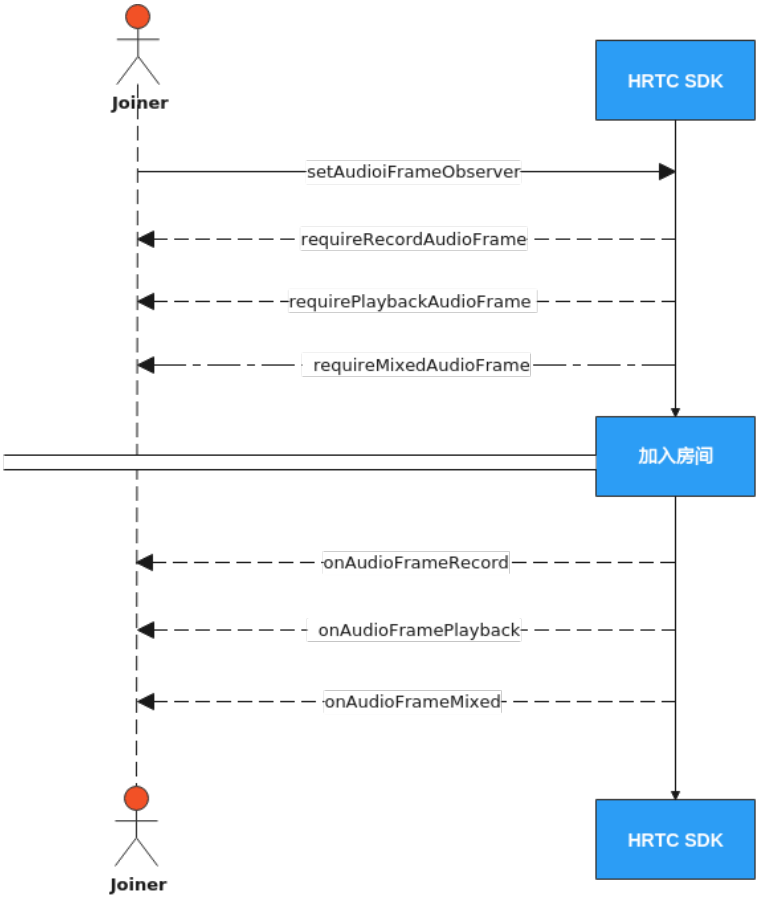
对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行音频前处理，然后发送给远端。也可以进行音频后处理。

注意事项

当前只支持PCM数据格式处理。

接口调用流程



注册音频前后处理

获取HWRtcEngine的HWRtcMediaEngine对象。

```
[HWRtcEngine sharedEngine].mediaEngine
```

注册音频前后处理

```
[[HWRtcEngine sharedEngine].mediaEngine setAudioFrameObserver:self];
```

每次入会都需要重新注册。取消注册，则传nil。

self（当前类）要签署HWRtcMediaEngineAudioDelegate。

实现如下回调：

```
requireRecordAudioFrame:
requirePlaybackAudioFrame:
requireMixedAudioFrame:
```

通过回调的返回值来决定对应音频帧的处理是否生效。

实现

```
onAudioFramePlayback:
onAudioFrameMixed:
onAudioFrameRecord:
```

回调，从回调中获取音频帧并进行处理。

注意：所有回调的返回值为false，说明对音频帧的处理无效。

```
/// 音频前后处理
- (BOOL)onAudioFramePlayback:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
    // 本地音频数据
    return YES;
}

- (BOOL)onAudioFrameMixed:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
// 混音数据
    return YES;
}

- (BOOL)onAudioFrameRecord:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
    // 远端音频数据
    return YES;
}

- (BOOL)requireRecordAudioFrame {
    // 返回值决定是否远端音频数据生效
    return YES;
}

- (BOOL)requirePlaybackAudioFrame {
// 返回值决定是否本地音频数据生效
    return YES;
}

-(BOOL)requireMixedAudioFrame {
    // 返回值决定是否混音数据生效
    return YES;
}
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setAudioFrameObserver

requireRecordAudioFrame

requirePlaybackAudioFrame

requireMixedAudioFrame

onPlaybackExternalAudioFrame

onAudioFramePlayback

onAudioFrameRecord

onAudioFrameMixed

2.2.7 音频自采集和音频自渲染

功能描述

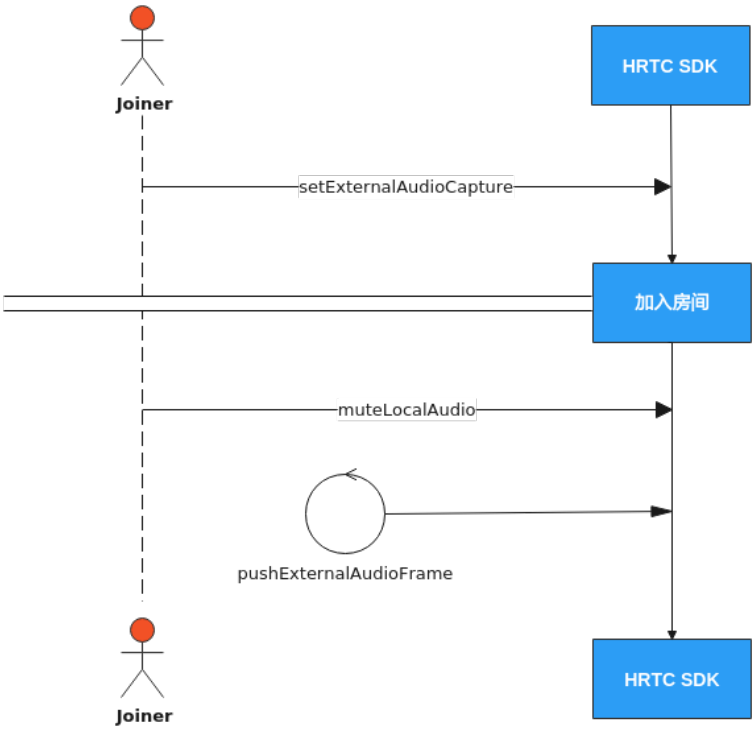
实时音频传输过程中，HWRtcEngine SDK通常会启动默认的音频模块进行采集和渲染。在以下场景中，您可能会发现默认的音频模块无法满足开发需求：

- app中已有自己的音频模块。
- 需要使用自定义的采集或播放处理。
- 某些音频采集设备被系统独占。

注意事项

当前只支持PCM数据格式处理。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalAudioCapture**

   加入房间前调用此接口打开自采集功能。

   ```
   [_rtcEngine setExternalAudioCapture:YES
                                sampleRate:16000
                                  channels:1]
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **开启音频流发送**

   调用接口muteLocalAudio开启音频流发送。

   ```
   [_rtcEngine  muteLocalAudio:NO];
   ```

4. **定时推送音频数据帧**

   加入房间成功回调后，每隔10ms定时调用pushExternalAudioFrame接口推送外部音频数据。

   音频输入数据大小：10 * sampleRate * channels * 16 / 8 / 1000 字节，其中的sampleRate和channels是前面调用的setExternalAudioCapture里传入的采样率和声道数参数。

```
// audioData  获取的音频数据
 [_rtcEngine pushExternalAudioFrame: audioData];
```

根据帧率循环调用pushExternalAudioFrame方法往SDK推送数据。

5. **音频自渲染**

   暂不支持音频自渲染功能。

API 参考

setExternalAudioCapture

muteLocalAudio

pushExternalAudioFrame

2.2.8 原始视频数据（视频前后处理）

功能描述

音视频传输过程中，我们可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

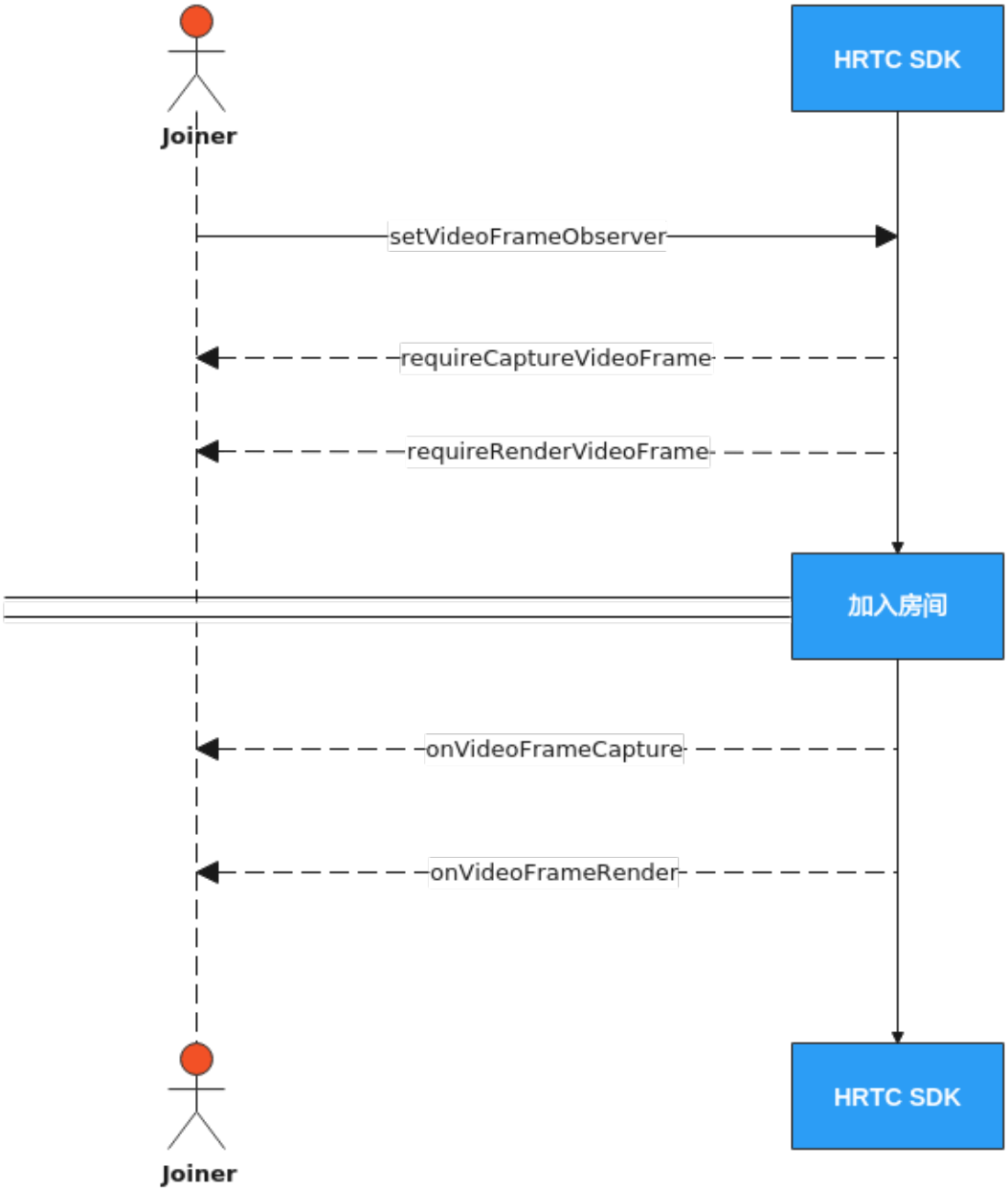
对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行视频前处理，然后发送给远端。也可以进行视频后处理。

注意事项

当前只支持YUV420数据格式处理。

接口调用流程



注册视频前后处理

获取HWRtcEngine的HWRtcMediaEngine对象。

```
[HWRtcEngine sharedEngine].mediaEngine
```

注册视频前后处理

```
[[HWRtcEngine sharedEngine].mediaEngine setVideoFrameObserver:self];
```

每次入会都需要重新注册。取消注册，则传nil。

self（当前类）要签署HWRtcMediaEngineVideoDelegate。

和实现

```
requireCaptureVideoFrame:
```

```
requireRenderVideoFrame:
```

回调，通过回调的返回值来决定对应视频帧的处理是否生效。

实现

```
onVideoFrameCapture:
onVideoFrameRender:
```

回调，从回调中获取视频帧并进行处理。

注意：所有回调的返回值为false，说明对视频帧的处理无效。

```
/// 视频前处理
- (BOOL)onVideoFrameCapture:(HWRtcVideoFrame* _Nonnull)videoFrame {
    return YES;
}

//后处理
- (BOOL)onVideoFrameRender:(NSString *_Nonnull)userid
                videoFrame:(HWRtcVideoFrame* _Nonnull)videoFrame
 {
    return YES;
}

- (BOOL)requireCaptureVideoFrame {
    return YES;
}

- (BOOL)requireRenderVideoFrame {
    Return YES;
}
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setVideoFrameObserver

requireCaptureVideoFrame

requireRenderVideoFrame

onVideoFrameCapture

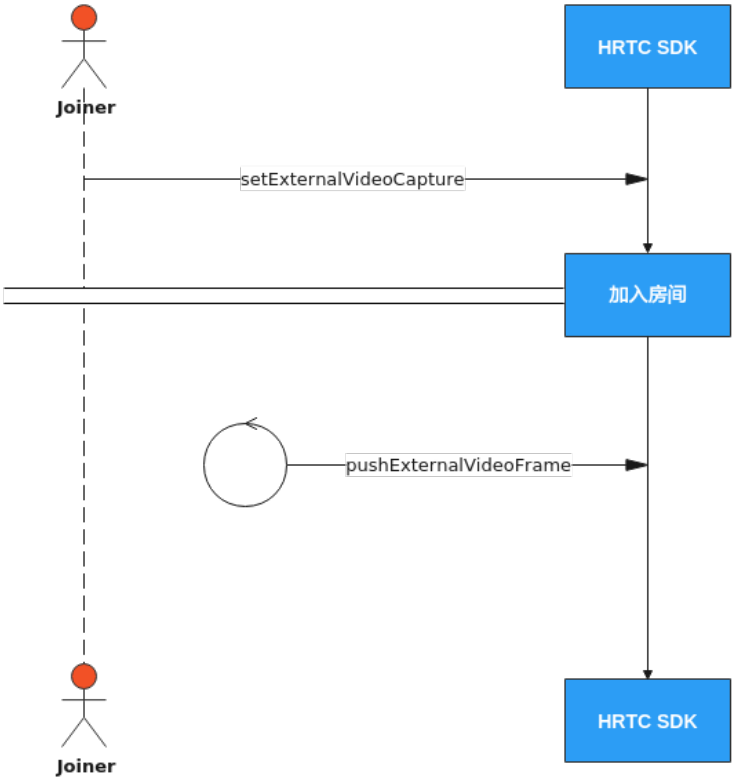
onVideoFrameRender

2.2.9 自定义视频采集

功能描述

如果您需要使用自定义的美颜库或有前处理库，则需要自己采集和处理摄像头拍摄画面，您可以通过SparkRTC SDK的setExternalVideoCapture接口开启 自采集功能。然后使用pushExternalVideoFrame接口推送外部视频数据到SparkRTC SDK播放。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalVideoCapture**

   加入房间前调用此接口打开视频自采集功能。一旦开启后，将无法切换。

   ```
   //开启自采集功能
   [rtcEngine setExternalVideoCapture:YES];
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **定时推送视频数据帧**

   加入房间成功回调后，上层应用需要定时调用pushExternalVideoFrame接口推送外部视频数据，每1/帧率调用一次。

   ```
   //初始化一个HWRtcVideoFrame对象，参数以实际为主，传rtcEngine
   HWRtcVideoFrame *rtcVideoFrame = [[HWRtcVideoFrame alloc] init];
    rtcVideoFrame.format =HWRtcVideoImageFormatYUV420P;
    rtcVideoFrame.width = width;
    rtcVideoFrame.height = height;
    rtcVideoFrame.data = data;
    rtcVideoFrame.dataLen = data.length;
    [_rtcEngine pushExternalVideoFrame:rtcVideoFrame];
   ```

API 参考

setExternalVideoCapture

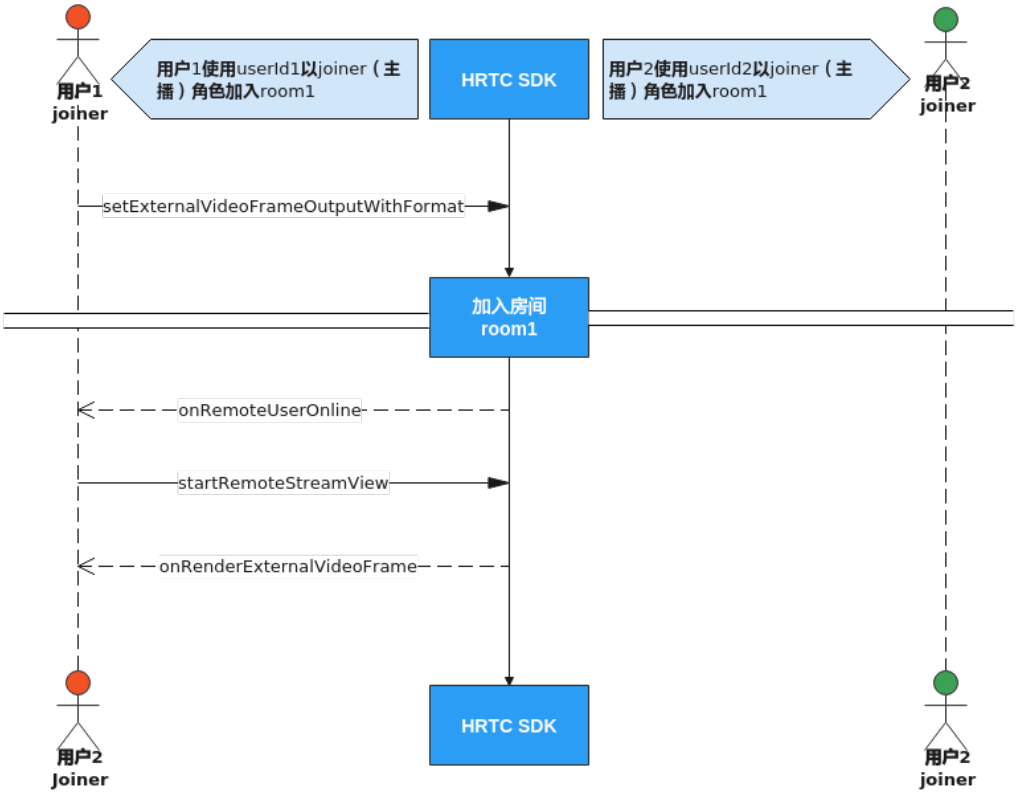
pushExternalVideoFrame

2.2.10 自定义视频渲染

功能描述

实时音视频传输过程中，上层应用可以不用SDK默认的渲染功能，选择对视频帧数据进行自定义渲染。

接口调用流程



实现过程

1. **加入房间前开启视频自渲染**

   加入房间前调用接口setExternalVideoFrameOutputWithFormat 打开视频自渲染功能。

   ```
   //开启远端或者本地的视频自渲染
   HwRtcImageBufferFormat * fileFormat = [[HwRtcImageBufferFormat alloc] init];
    fileFormat.format = HWRtcVideoImageFormatYUV420P;
    fileFormat.bufferType = HWRtcVideoImageBufferByteArray;
   [rtcEngine setExternalVideoFrameOutputWithFormat:fileFormat remoteEnable:YES localEnable:YES];
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **渲染远端视频流**

   当远端用户加入房间后会触发onRemoteUserOnline回调，在该回调中使用startRemoteStreamView方法开启接收远端用户的视频流，如果不开启，无法渲染远端视频流。

   ```
   //远端用户加入房间后触发的回调
   - (void)onRemoteUserOnline:(NSString * _Nonnull)roomId
   ```

```
            userId:(NSString * _Nonnull)userId
          userName:(NSString * _Nonnull)userName
{
//创建视频画布对象
HWRtcVideoCanvas *canvas = [[HWRtcVideoCanvas alloc] init];
canvas.uid = @”remote userId”;
canvas.view = [[HWRtcView alloc] initWitFrame:(0,0,100,100)];

// 开启远端视频流接收
 // @param remote 参考 HWRtcVideoCanvas
 // @param streamType  视频分辨率
 / /@param disableAdjustRes: 禁用分辨率自适应，默认关闭。
[rtcEngine startRemoteStreamView:canvas streamType:HWRtcStreamTypeHD disableAdjustRes:YES];
}
```

4. **触发回调**

   加入房间后sdk会根据前面的参数设置调用onRenderExternalVideoFrame回调函数上报本地和远端视频帧数据给上层应用处理。

```
//自渲染回调
- (void)onRenderExternalVideoFrame:(NSString *_Nonnull)roomid
            meidaDirection:(HWRtcMediaDirection)meidaDirection
               videoFrame:(HWRtcVideoFrame * _Nonnull)videoFrame
{
//获取videoFrame，进行数据处理
}
```

API 参考

setExternalVideoFrameOutputWithFormat

onRemoteUserOnline

startRemoteStreamView

onRenderExternalVideoFrame

2.2.11 加入多频道（跨房）

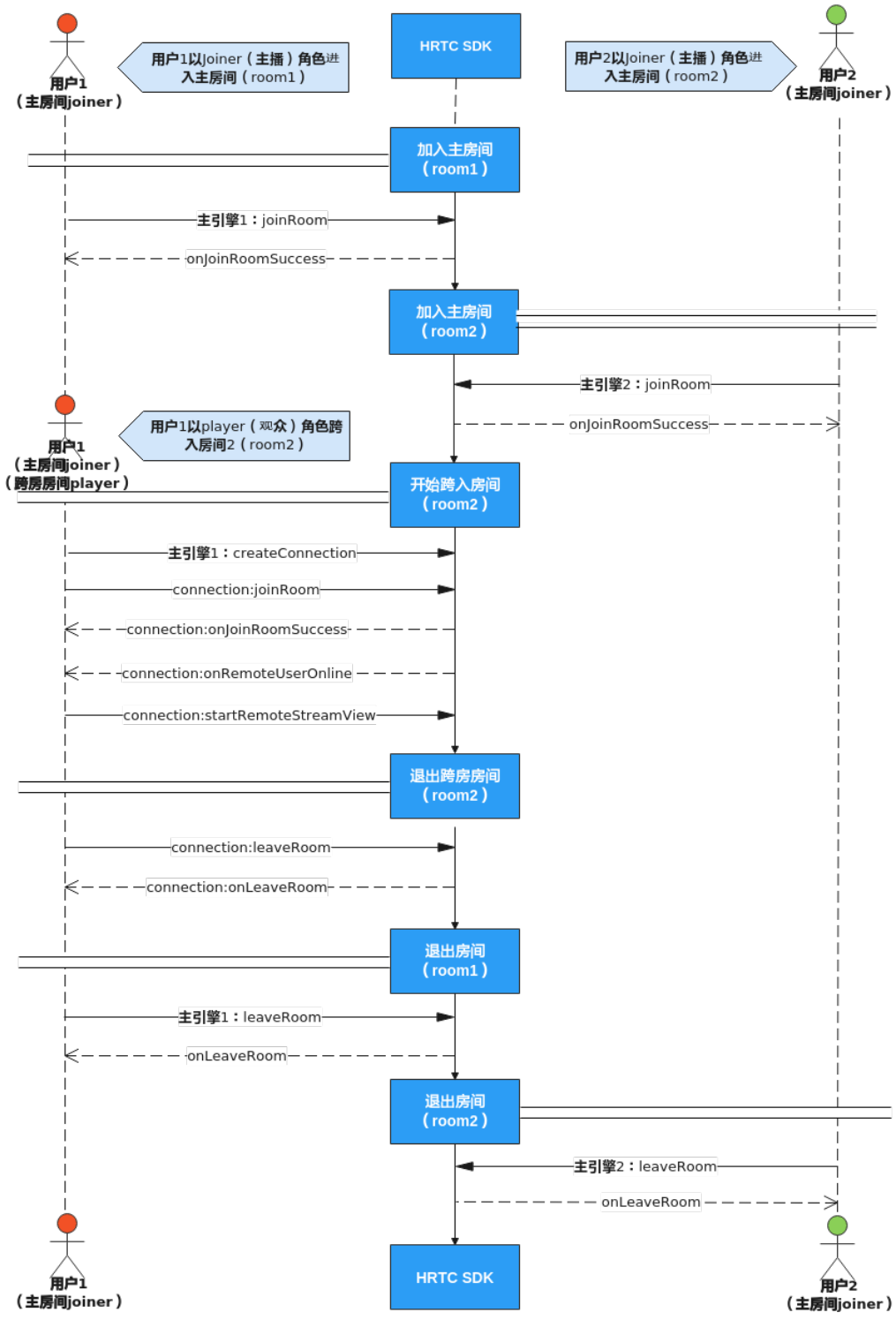
功能描述

跨房间连麦，指主播的媒体流可以同时转发进多个（目前最多支持四个）房间频道，实现主播跨频道与其他主播实时互动的场景。房间中的所有主播可以看见彼此，房间中的观众可以看到所有主播。

注意事项

- 同一时间最多只能创建4个连接对象，每个连接对象对应的房间ID必须互不相同。
- 如果使用connection对象加入房间，则加入房间的房间ID不能和已创建连接对象对应的房间ID相同。
- 同一时间只能以一个JOINER角色加入某一个房间。如果本端在其他房间里的角色是HWRtcRoleJoiner，则需要调用该房间的setUserRole方法将本端在该房间的角色切换为HWRtcRolePlayer后才能以HWRtcRoleJoiner跨入其他房间。如果本端用户是以HWRtcRolePlayer角色加入此跨房房间，则需要调用跨房连接的setUserRole方法将本端的角色切换为HWRtcRoleJoiner后才能发送音频流和视频流。

接口调用流程



实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入主房间。

2. **创建跨房连接**

当收到远端用户加入房间后会触发connection:onRemoteUserOnline跨房回调，在该回调内用对应远端用户的跨房连接调用startRemoteStreamView方法设置远端窗口并开启收流。

示例代码如下：

```
//创建跨房引擎，room id具有唯一性，不可重复，设置引擎代理
HWRtcConnection *connection = [rtcEngine createConnection:@”room id”];
connection.delegate = self;
```

3. **调用跨房连接的joinRoom加入跨房房间**

调用跨房连接的joinRoom接口加入房间，其中HWRtcUserInfo的signature鉴权签名字符串需要填入正确的计算值，计算方法请参考**接入鉴权**。

如果本端需要发送音频或者视频流给跨房房间里的远端用户，则用户的role参数需要设为HWRtcRoleJoiner，否则可以设为HWRtcRolePlayer，只收流不发流。

注意：不能同时以**JOINER**角色加入多个房间。

加入跨房房间成功后会收到跨房回调指针的onJoinRoomSuccess通知，加入房间失败会收到跨房回调指针的onJoinRoomFailure通知。

示例代码如下：

```
//创建HWRtcUserInfo对象
HWRtcJoinParam  *joinRoomParam = [[HWRtcJoinParam alloc] init];
joinRoomParam.autoSubscribeAudio = YES;
joinRoomParam.autoSubscribeVideo = YES;
joinRoomParam.userId = "userId";
joinRoomParam.userName = "userName";
joinRoomParam.authorization = authorization;//authorization：鉴权信息，具体生成方法请参见接入鉴权。与加入房间的authorization相同
joinRoomParam.ctime = time;//与加入房间的time相同
joinRoomParam.roomId = "roomId";
joinRoomParam.userRole = HWRtcRolePlayer;
joinRoomParam.sfuType = 0;
joinRoomParam.scenario = 0;
//通过跨房引擎进入房间
[connection joinRoom:joinRoomParam];
```

4. **收远端用户的视频流**

收到远端用户加入房间后会收到跨房回调connection:onRemoteUserOnline，可以调用该远端用户对应的跨房连接的startRemoteStreamView设置远端窗口并开启收流。

示例代码如下：

```
//远端用户加入房间后会收到跨房回调
- (void)connection:(HWRtcConnection *)connection
     onRemoteUserOnline:(NSString * _Nonnull)userId
          userName:(NSString * _Nonnull)userName
{
    //创建视频画布对象
    HWRtcVideoCanvas *canvas = [[HWRtcVideoCanvas alloc] init];
    canvas.uid = @” remote userId”;
    canvas.view = [[HWRtcView alloc] initWitFrame:(0,0,100,100)];
    //设置远端窗口并开启收流
    [connection startRemoteStreamView:canvas streamType:HWRtcStreamTypeHD
disableAdjustRes:YES];
}
```

5. **退出跨房房间**

跨房结束后，调用跨房连接的leaveRoom接口跨房房间。

示例代码如下：

```
//退出跨房房间
[connection leaveRoom];
```


API 参考

createConnection

joinRoom

connection:onJoinSuccess

connection:onJoinRoomFailure

connection:onRemoteUserOnline

startRemoteStreamView

leaveRoom

2.3 实现音视频通话（MAC）

2.3.1 环境准备

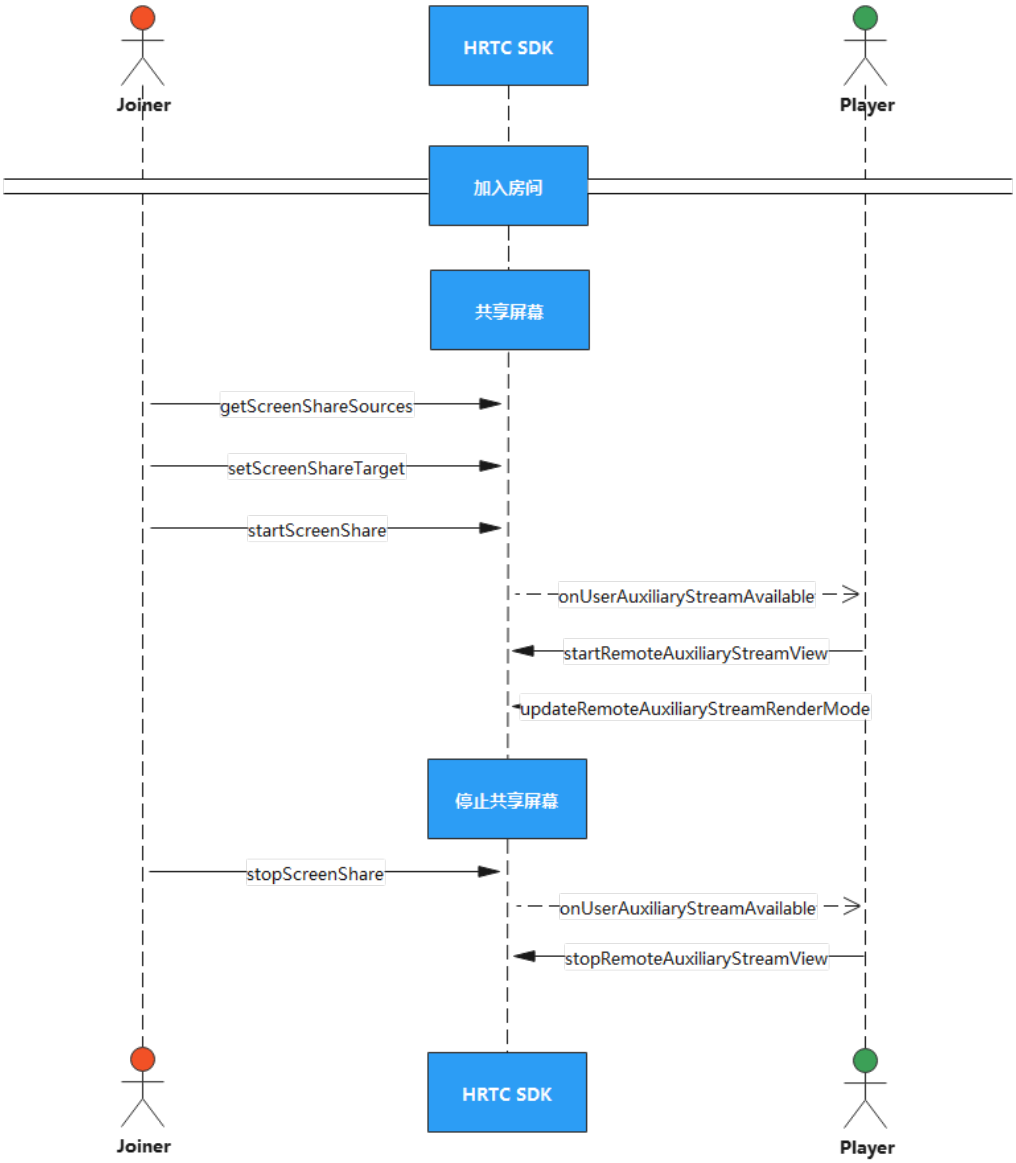
详情请参考**开发前准备**。

2.3.2 屏幕分享

功能描述

屏幕共享用于在音视频会议中，把一个与会者的屏幕内容，以视频的方式分享给其他与会者。屏幕共享可以共享整个桌面，也可以共享某一个程序窗口。共享程序窗口的时候还可以指定共享该程序窗口的矩形区域。

接口调用流程



实现屏幕共享

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **共享桌面**

   加入房间后可以调用getScreenShareSources获取屏幕共享对象列表，然后遍历找到其中HRTCScreenShareSourceInfo.type为HWRtcScreenShareTypeDesktop的对象，该对象就是屏幕对象，再调用setScreenShareTarget设置共享桌面参数。桌面共享不支持区域共享。

   然后调用startScreenShare开始共享。

   示例代码如下：

   ```
   //获取引擎
   rtcEngine = [HWRtcEngine sharedEngine];
   // 定义屏幕共享区域参数
   HWRtcScreenShareParam *param = [[HWRtcScreenShareParam alloc] init];
   ```

```
 param.type = HWRtcScreenShareTypeDesktop;
 param.bCaptureMouse = NO;
 param.viewID = 0;
 param.rect = rect;
// 获取屏幕共享列表，包含桌面共享对象
NSArray* sourceList = [self.rtcEngine getScreenShareSources];
for (int i= 0; i< sourceList.count; i++) {
    HRTCScreenShareSourceInfo sourceInfo = sourceList[i];
    if (sourceInfo.type == HWRtcScreenShareTypeDesktop) { //如果是桌面共享对象
        // 调用设置屏幕共享接口
        [rtcEngine setScreenShareTarget:param];
        // 开启屏幕共享
        [rtcEngine startScreenShare];
        break;
    }
}
```

3. **共享程序**

   加入房间后也可以调用getScreenShareSources来获取可以共享的程序窗口列表，然后展示在界面上供用户选择。

   选择好需要共享的程序窗口后，调用setScreenShareTarget来设置要共享的程序窗口，其中HRTCScreenShareSourceInfo.type设为HRTC_WINDOW_CAPTURE，HRTCSrceenCaptureOptionalInfo参数用于指定共享的程序窗口的子矩形区域，矩形区域以共享的程序窗口的左上角为原点(0,0)，最小96*92 最大1920*1080。如果需要共享该程序的完整窗口，则**HRTCSrceenCaptureOptionalInfo**参数的矩形区域全填为0。

   然后调用startScreenShare开始共享。

   示例代码如下：

```
//获取共享程序窗口列表
rtcEngine = [HWRtcEngine sharedEngine];
NSArray* sourceList = [self.rtcEngine getScreenShareSources];
for (int i= 0; i< sourceList.count; i++) {
    HRTCScreenShareSourceInfo sourceInfo = sourceList[i];
   //将sourceInfo加入列表展示到界面上供用户选择
}

//用户选择共享的程序后
HWRtcScreenShareParam *param = [[HWRtcScreenShareParam alloc] init];
 param.type = HWRtcScreenShareTypeDesktop;
 param.bCaptureMouse = NO;
 param.viewID = viewID;
 param.rect = rect;
// 调用设置屏幕共享接口
[rtcEngine setScreenShareTarget:param];
// 开启屏幕共享
[rtcEngine startScreenShare];
```

4. **接收远端用户的共享流**

   收到远端用户开启共享流通知onUserAuxiliaryStreamAvailable后，可以调用startRemoteAuxiliaryStreamView来设置远端用户的共享流的窗口句柄并开始选看。

   还可以再调用updateRemoteAuxiliaryStreamRenderMode设置窗口显示共享流的方式。

   示例代码如下：

```
[rtcEngine startRemoteAuxiliaryStreamView:canvas streamType:HWRtcStreamTypeLD];
```

5. **停止屏幕共享**

   屏幕共享结束时，可以调用stopScreenShare停止屏幕共享。

   示例代码如下：

```
[rtcEngine stopScreenShare];
```

6. **停止接收远端用户的屏幕共享流**

   收到onUserAuxiliaryStreamAvailable消息后，如果选看的远端屏幕共享流不可用，或者收到远端用户下线通知onRemoteUserOffline，则接收端必须调用stopRemoteAuxiliaryStreamView关闭共享流窗口视图。

   如果接收端需要主动停止接收远端用户的共享流，也需要调用stopRemoteAuxiliaryStreamView接口停止接收共享流。

   示例代码如下：

   ```
   [rtcEngine stopRemoteAuxiliaryStreamView:userId];
   ```

API 参考

startRemoteAuxiliaryStreamView

stopRemoteAuxiliaryStreamView

startScreenShare

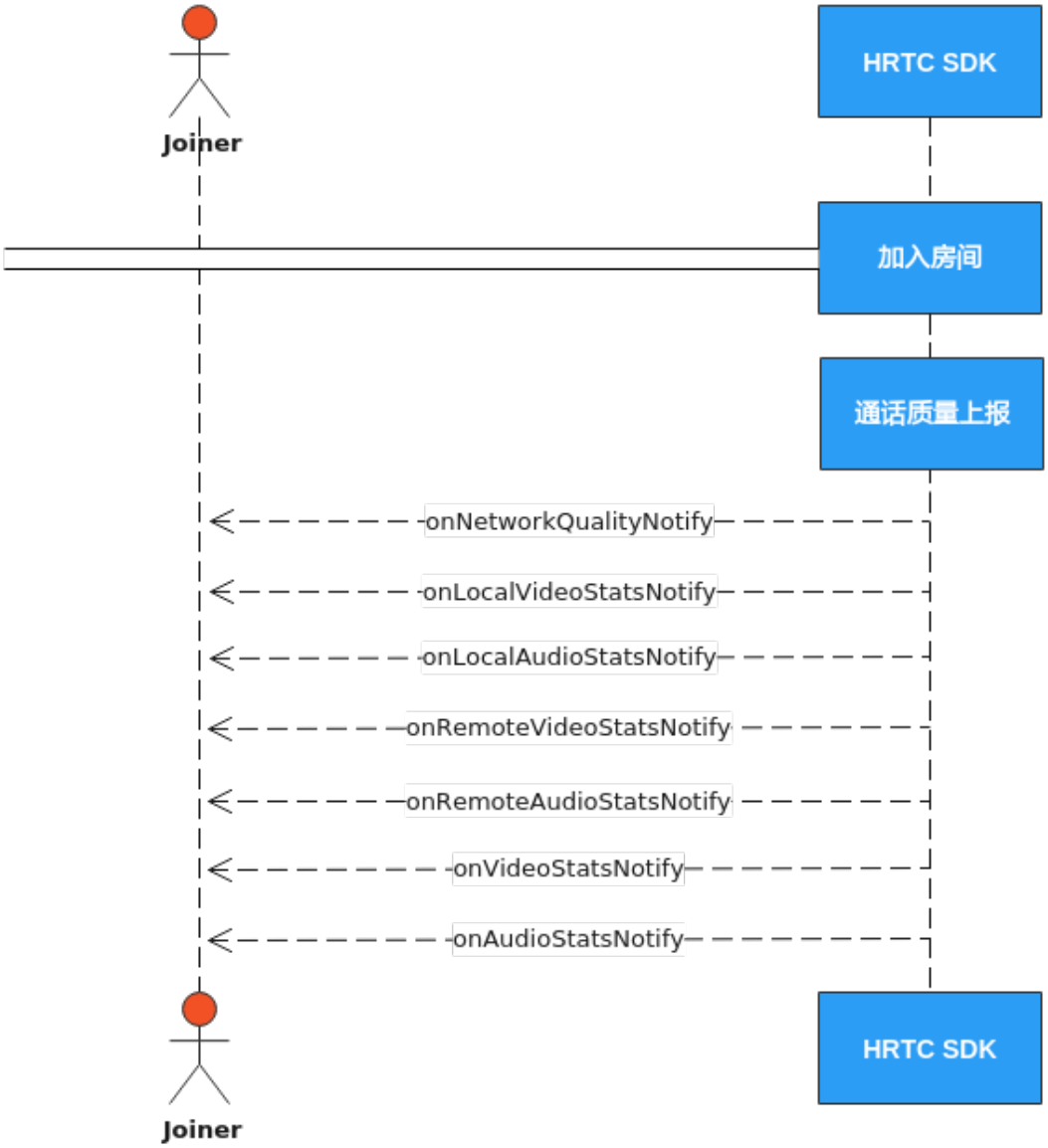
stopScreenShare

2.3.3 通话质量监测

功能描述

加入频道后，SDK会每隔2秒自动触发通话质量相关的回调，上报当前通话的网络质量、本地和远端的音视频统计信息。

上报接口



通话质量上报

onNetworkQualityNotify回调上报当前通话中每个入会者的上下行网络质量。默认开启，每2s上报一次。

```
- (void)onNetworkQualityNotify:(NSArray <HWRtcQualityInfo *> * _Nullable)upStreamQualityArray
      downStreamQualityInfo:(NSArray <HWRtcQualityInfo *> * _Nullable)downStreamQualityArray
{
    dispatch_async(dispatch_get_main_queue(),^{
        for (HWRtcQualityInfo *info in upStreamQualityArray) {
            //本地网络质量信息数据处理
        }
        for (HWRtcQualityInfo *info in downStreamQualityArray) {
            //远端网络质量信息数据处理
        }
    });
}
```

本地音频流统计信息报告

onLocalAudioStatsNotify回调上报本地设备发送音频流的统计信息。您可以了解到当前通话声道数（单声道或双声道）、发送音频的采样率、码率、比特率、丢包率、延时和抖动等。

```
- (void)onLocalAudioStatsNotify:(NSArray <HWRtcLocalAudioStats *> * _Nullable)localAudioStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcLocalAudioStats *audioStatsInfo in localAudioStatsArray) {
            //本地音频信息数据处理
        }
    });
}
```

远端音频流统计信息报告

onRemoteAudioStatsNotify回调上报当前通话中每个远端用户音频流的统计信息。您可以了解到每个远端用户发送的音频流的采样率、声道数、码率、丢包率、延时、抖动和卡顿时长等一些信息。

```
- (void)onRemoteAudioStatsNotify:(NSArray <HWRtcRemoteAudioStats *> *
_Nullable)remoteAudioStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcRemoteAudioStats *audioStatsInfo in remoteAudioStatsArray) {
          //远端音频信息数据处理
        }
    });
}
```

本地视频流统计信息上报

onLocalVideoStatsNotify回调向您报告当前本地视频流统计信息，包括帧率、码率、延时、抖动等信息。

```
- (void)onLocalVideoStatsNotify:(NSArray <HWRtcLocalVideoStats *> * _Nullable)localVideoStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcLocalVideoStats *videoStatsInfo in localVideoStatsArray) {
          //本地视频信息数据处理
        }
    });
}
```

远端视频流状态监控

onRemoteVideoStatsNotify回调向您报告当前远端视频流统计信息，包括帧率、码率、延时、抖动和卡顿时长等信息。

```
-(void)onRemoteVideoStatsNotify:(NSArray <HWRtcRemoteVideoStats *> *
_Nullable)remoteVideoStatsArray{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcRemoteVideoStats *videoStatsInfo in remoteVideoStatsArray) {
          //处理远端视频信息
        });
    }
}
```

视频流状态监控

onVideoStatsNotify回调上报视频流的状态，包括本地上行视频流和远端用户的下行视频流状态。

```
- (void)onVideoStatsNotify:(NSArray<HWRtcVideoStatsInfo *> *)videoStatsArray remoteVideoInfo:
(NSArray<HWRtcVideoStatsInfo *> *)remoteVideoStatsInfos
```

```
{
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcVideoStatsInfo *videoStatsInfo in videoStatsArray) {
            //本地视频信息数据处理
        }
        for (HWRtcVideoStatsInfo *remoteVideoStatsInfo in videoStatsArray) {
            //远端视频信息数据处理
        }
    });
}
```

音频流状态监控

onAudioStatsNotify回调上报音频流的状态，包括本地上行音频流和远端用户的下行音频流状态。

```
- (void)onAudioStatsNotify:(NSArray<HWRtcAudioStatsInfo *> *)audioStatsArray remoteAudioInfo:
(NSArray<HWRtcAudioStatsInfo *> *)remoteAudioStatsInfos
{https://support.huaweicloud.com/csdk-rtc/rtc_05_0167.html
    dispatch_async(dispatch_get_main_queue(), ^{
        for (HWRtcAudioStatsInfo *audioStatsInfo in audioStatsArray) {
            //本地音频信息数据处理
        }
        for (HWRtcAudioStatsInfo *remoteAudioStatsInfo in remoteAudioStatsInfos) {
            //远端音频信息数据处理
        }
    });
}
```

API 参考

onNetworkQualityNotify

onLocalVideoStatsNotify

onLocalAudioStatsNotify

onRemoteVideoStatsNotify

onRemoteAudioStatsNotify

onVideoStatsNotify

onAudioStatsNotify

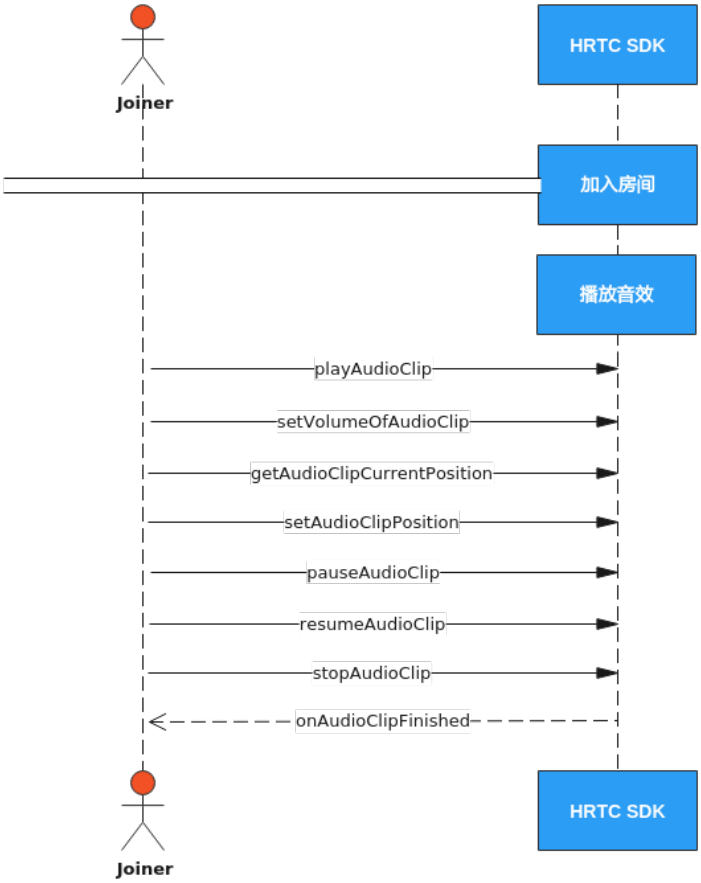
2.3.4 播放音效

功能描述

用户可以同时播放多个音效文件，给自己和其他与会者听，用于烘托气氛。

支持播放wav、pcm和单声道mp3音频文件，支持本地或在线文件路径。

接口调用流程



实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音效文件**

   调用playAudioClip接口可以播放一个音效文件。可以同时播放多个音效文件，不同音效文件用不同的soundId参数进行区别。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine playAudioClip:soundId filePath: [[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]
    loop:1 pitch:0.00 pan:0.00 gain:100 publish:1 startPos:0];
   ```

3. **设置音效文件音量**

   音效文件播放过程中，可以调用setVolumeOfAudioClip设置音效文件播放音量。

   音量大小，范围为0-100。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine setVolumeOfAudioClip:soundId volume:50];
   ```

4. **获取音效文件总时长**

   播放过程中可以调用getAudioClipDuration获取音效文件总时长，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int duration = [rtcEngine getAudioClipDuration: [[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]];
   //根据当前总时长刷新界面进度条
   ```

5. **获取音效文件播放位置**

   播放过程中可以调用getAudioClipCurrentPosition获取音效文件播放位置，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int pos = [rtcEngine getAudioClipCurrentPosition:soundId];
   //根据当前播放位置刷新界面进度条
   ```

6. **设置音效文件播放位置**

   播放过程中可以调用setAudioClipPosition设置音效文件播放位置，可用于通过拖动进度条改变音效文件的播放位置。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine setAudioClipPosition:soundId pos:50];
   ```

7. **暂停播放音效文件**

   调用pauseAudioClip接口可以暂停播放一个音效文件。调用pauseAllAudioClips暂停播放所有正在播放的音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine pauseAudioClip:soundId];
   ```

8. **恢复播放暂停的音效文件**

   音频文件暂停播放后，可以调用resumeAudioClip接口可以恢复播放之前暂停的音效文件。或者调用resumeAllAudioClips恢复播放所有暂停的音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine resumeAudioClip:soundId];
   ```

9. **停止播放音效文件**

   调用stopAudioClip接口停止播放一个音效文件，或者调用stopAllAudioClips接口停止播放所有音效文件。

   ```
   int soundId = 0; //音效文件ID
   [rtcEngine stopAudioClip:soundId];
   ```

10. **音效文件播放结束回调**

    音效文件播放结束后，sdk会触发onAudioClipFinished回调通知上层应用。

    ```
    - (void)onAudioClipFinished:(NSInteger)soundId
    {
      //收到音效文件播放结束通知后，可以刷新界面，比如恢复播放前初始界面状态。
    }
    ```

API 参考

playAudioClip

setVolumeOfAudioClip

getAudioClipDuration

getAudioClipCurrentPosition

setAudioClipPosition

pauseAudioClip

pauseAllAudioClips

resumeAudioClip

resumeAllAudioClips

stopAudioClip

stopAllAudioClips

onAudioClipFinished

2.3.5 播放音乐

功能描述

混音是将音乐文件和麦克风音频混合，一般用于播放时长较长的背景音乐或者伴奏。同一时间只能播放一个音乐文件。可以在本地播放，也可以同时播放给其他与会者听。

支持播放wav、pcm和单声道mp3音频文件。可以播放本地或在线音乐文件。

接口调用流程

实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音乐文件**

   调用startAudioFile接口可以播放一个音乐文件。同一时刻只能播放一个音乐文件。

```
[rtcEngine startAudioFile:[[NSBundle mainBundle] pathForResource:@"test.mp3" ofType:@""]
publish:1 cycle:1 replace:0];
```

3. **设置音乐文件音量**

   音乐文件播放过程中，可以调用adjustAudioFileVolume设置音乐文件播放音量。

```
[rtcEngine adjustAudioFileVolume:50];
```

4. **获取音乐文件总时长和播放位置**

   音乐文件打开成功后，sdk会触发一次onAudioMixStateChangedNotify回调，传入state参数为HWRtcAudioFileOpenCompleted，此时可以调用getAudioFileDuration获取音乐文件总时长，可用以刷新界面进度条的总时长。

   音乐文件在播放过程中，sdk每秒会触发一次onAudioMixStateChangedNotify回调，其中的state参数为HWRtcAudioFilePositionUpdate，value参数就是当前的音乐文件播放进度，以毫秒为单位，可以在此回调中刷新界面的播放进度条。

   播放过程中也可以调用getAudioFileCurrentPosition获取音乐文件当前播放位置，可用于刷新界面的播放进度条。

```
- (void)onAudioMixStateChangedNotify:(HWRtcAudioFileState)state reason:
(HWRtcAudioFileReason)reason value:(NSUInteger)value{
    switch (reason) {
        case HWRtcAudioFileReasonNone:
        {
            switch (state) {
                case HWRtcAudioFileOpenCompleted:
                {
                    //获取总时长，并刷新进度条
                    int mixDuration = [self.rtcEngine getAudioFileDuration];
                }
                    break;
                case HWRtcAudioFilePositionUpdate:
                {
                    //使用value参数值刷新界面的播放进度条
                }
                    break;
                case HWRtcAudioFilePlaying:
                {
                    //刷新界面
                }
                    break;
                case HWRtcAudioFilePaused:
                {
                    //刷新界面
                }
                    break;
                case HWRtcAudioFileStopped:
                {
                    //刷新界面
                }
                    break;
                default:
                    break;
            }
        }
            break;
        default:
        {
            //错误信息提示
        }
            break;
    }
}
```

5. **设置音乐文件播放位置**

   播放过程中可以调用setAudioFilePosition设置音乐文件播放位置，可用于通过拖动进度条改变音乐文件的播放位置。

```
[rtcEngine setAudioFilePosition:50];
```

6. **暂停播放音乐文件**

   调用pauseAudioFile接口可以暂停播放一个音乐文件。

   ```
   [rtcEngine pauseAudioFile];
   ```

7. **恢复播放暂停的音乐文件**

   音乐文件暂停播放后，可以调用resumeAudioFile接口恢复播放。

   ```
   [rtcEngine resumeAudioFile];
   ```

8. **停止播放音乐文件**

   调用stopAudioFile接口可以停止播放音乐文件。

   ```
   [rtcEngine stopAudioFile];
   ```

9. **音乐文件播放结束回调**

   音乐文件播放结束后，sdk会触发onAudioMixStateChangedNotify回调来通知上层应用。

   ```
   - (void)onAudioMixStateChangedNotify:(HWRtcAudioFileState)state reason:
   (HWRtcAudioFileReason)reason value:(NSUInteger)value{
       //收到音乐文件播放结束的通知，可以刷新界面，比如恢复到播放前初始界面状态
       if ( state == HWRtcAudioFilePlayCompleted ) {
       }
   }
   ```

API 参考

startAudioFile

adjustAudioFileVolume

getAudioFileDuration

getAudioFileCurrentPosition

setAudioFilePosition

pauseAudioFile

resumeAudioFile

stopAudioFile

onAudioMixStateChangedNotify

2.3.6 原始音频数据（音频前后处理）

功能描述

音视频传输过程中，可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

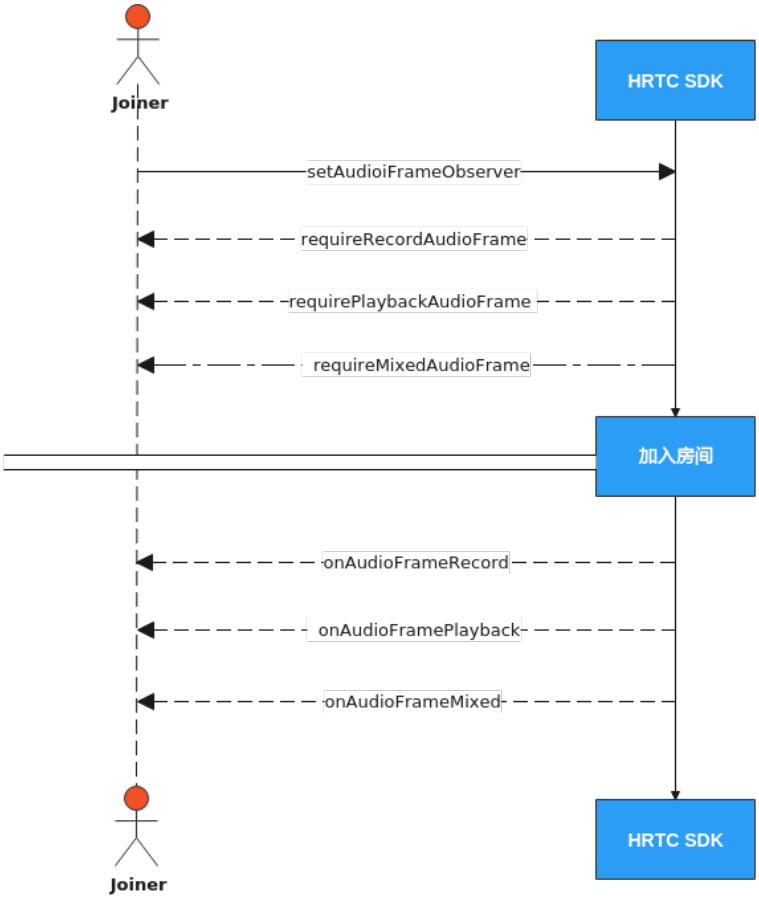
对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行音频前处理，然后发送给远端。也可以进行音频后处理。

注意事项

当前只支持PCM数据格式处理。

接口调用流程



注册音频前后处理

获取HWRtcEngine的HWRtcMediaEngine对象。

```
[HWRtcEngine sharedEngine].mediaEngine
```

注册音频前后处理

```
[[HWRtcEngine sharedEngine].mediaEngine setAudioFrameObserver:self];
```

每次入会都需要重新注册。取消注册，则传nil。

self（当前类）要签署HWRtcMediaEngineAudioDelegate。

实现如下回调：

```
requireRecordAudioFrame:
requirePlaybackAudioFrame:
requireMixedAudioFrame:
```

通过回调的返回值来决定对应音频帧的处理是否生效。

实现

```
onAudioFramePlayback:
onAudioFrameMixed:
onAudioFrameRecord:
```

回调，从回调中获取音频帧并进行处理。

注意：所有回调的返回值为false，说明对音频帧的处理无效。

```
/// 音频前后处理
- (BOOL)onAudioFramePlayback:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
    // 本地音频数据
    return YES;
}

- (BOOL)onAudioFrameMixed:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
// 混音数据
    return YES;
}

- (BOOL)onAudioFrameRecord:(HWRtcAudioFrame * _Nonnull)audioFrame
 {
    // 远端音频数据
    return YES;
}

- (BOOL)requireRecordAudioFrame {
    // 返回值决定是否远端音频数据生效
    return YES;
}

- (BOOL)requirePlaybackAudioFrame {
// 返回值决定是否本地音频数据生效
    return YES;
}

-(BOOL)requireMixedAudioFrame {
    // 返回值决定是否混音数据生效
    return YES;
}
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setAudioFrameObserver

requireRecordAudioFrame

requirePlaybackAudioFrame

requireMixedAudioFrame

onPlaybackExternalAudioFrame

onAudioFramePlayback

onAudioFrameRecord

onAudioFrameMixed

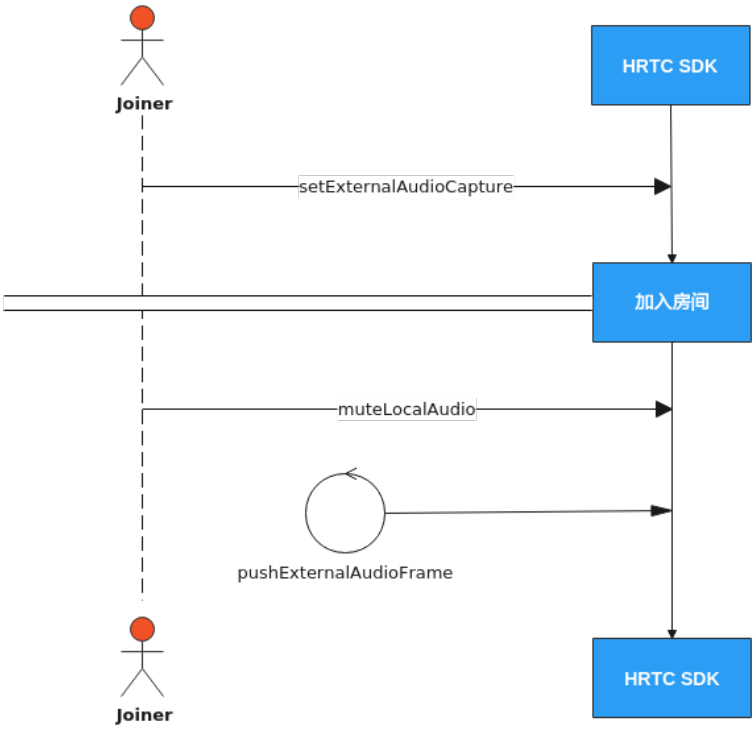
2.3.7 音频自采集和音频自渲染

功能描述

实时音频传输过程中，HWRtcEngine SDK通常会启动默认的音频模块进行采集和渲染。在以下场景中，您可能会发现默认的音频模块无法满足开发需求：

- 需要使用自定义的采集或播放处理。
- 某些音频采集设备被系统独占。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalAudioCapture**

   加入房间前调用此接口打开自采集功能。

   ```
   [_rtcEngine setExternalAudioCapture:YES
                                sampleRate:16000
                                  channels:1]
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **开启音频流发送**

   调用接口muteLocalAudio开启音频流发送。

   ```
   [_rtcEngine  muteLocalAudio:NO];
   ```

4. **定时推送音频数据帧**

   加入房间成功回调后，每隔10ms定时调用pushExternalAudioFrame接口推送外部音频数据。

   音频输入数据大小：10 * sampleRate * channels * 16 / 8 / 1000 字节，其中的sampleRate和channels是前面调用的setExternalAudioCapture里传入的采样率和声道数参数。

   ```
   // audioData  获取的音频数据
    [_rtcEngine pushExternalAudioFrame: audioData];
   ```

   根据帧率循环调用pushExternalAudioFrame方法往SDK推送数据。

5. **音频自渲染**

暂不支持音频自渲染功能。

注意：

当前只支持PCM数据格式处理

API 参考

setExternalAudioCapture

muteLocalAudio

pushExternalAudioFrame

2.3.8 原始视频数据（视频前后处理）

功能描述

音视频传输过程中，我们可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

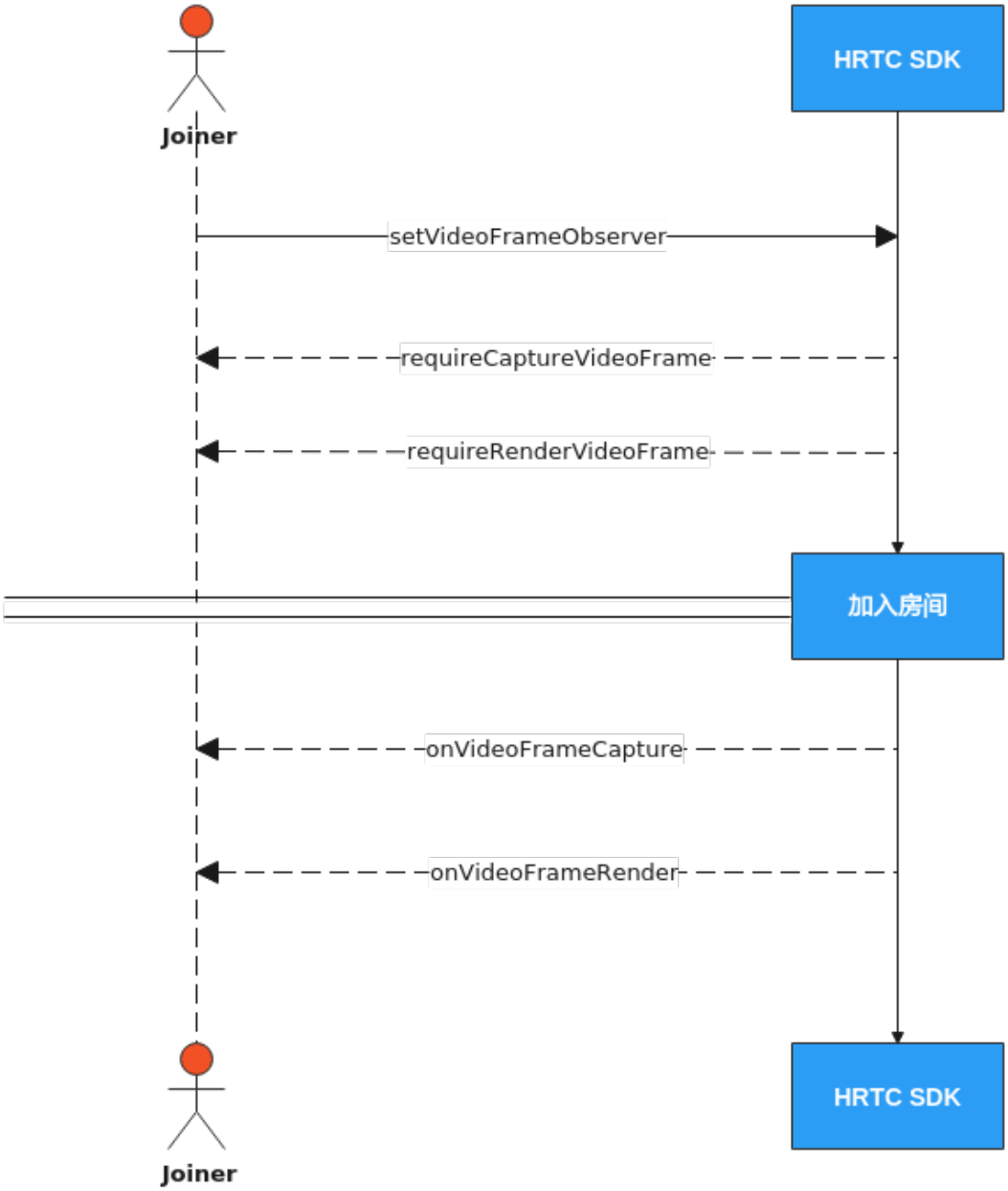
对于有自行处理音视频数据需求的场景，HWRtcEngine SDK 提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行视频前处理，然后发送给远端。也可以进行视频后处理。

注意事项

当前只支持YUV420数据格式处理。

接口调用流程



注册视频前后处理

获取HWRtcEngine的HWRtcMediaEngine对象。

```
[HWRtcEngine sharedEngine].mediaEngine
```

注册视频前后处理

```
[[HWRtcEngine sharedEngine].mediaEngine setVideoFrameObserver:self];
```

每次入会都需要重新注册。取消注册，则传nil。

self（当前类）要签署HWRtcMediaEngineVideoDelegate。

和实现

```
requireCaptureVideoFrame:
```

```
requireRenderVideoFrame:
```

回调，通过回调的返回值来决定对应视频帧的处理是否生效。

实现

```
onVideoFrameCapture:
onVideoFrameRender:
```

回调，从回调中获取视频帧并进行处理。

注意：所有回调的返回值为false，说明对视频帧的处理无效。

```
/// 视频前处理
- (BOOL)onVideoFrameCapture:(HWRtcVideoFrame* _Nonnull)videoFrame {
    return YES;
}

//后处理
- (BOOL)onVideoFrameRender:(NSString *_Nonnull)userid
                videoFrame:(HWRtcVideoFrame* _Nonnull)videoFrame
 {
    return YES;
}

- (BOOL)requireCaptureVideoFrame {
    return YES;
}

- (BOOL)requireRenderVideoFrame {
    Return YES;
}
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setVideoFrameObserver

requireCaptureVideoFrame

requireRenderVideoFrame

onVideoFrameCapture

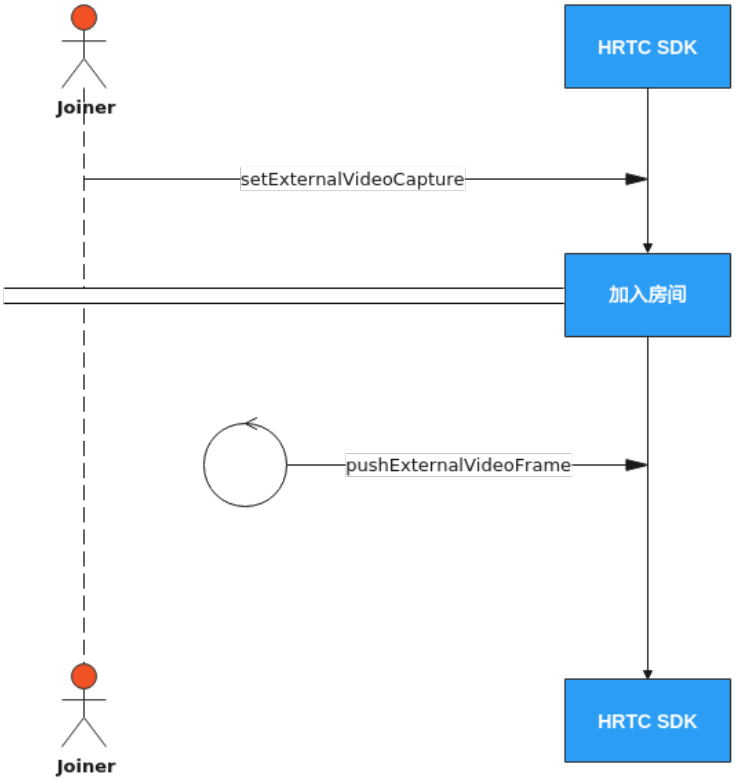
onVideoFrameRender

2.3.9 自定义视频采集

功能描述

如果您需要使用自定义的美颜库或有前处理库，则需要自己采集和处理摄像头拍摄画面，您可以通过SparkRTC SDK的setExternalVideoCapture接口开启 自采集功能。然后使用pushExternalVideoFrame接口推送外部视频数据到SparkRTC SDK播放。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalVideoCapture**

   加入房间前调用此接口打开视频自采集功能。一旦开启后，将无法切换。

   ```
   //开启自采集功能
   [rtcEngine setExternalVideoCapture:YES format:HWRtcVideoImageFormatYUV420P];
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **定时推送视频数据帧**

   加入房间成功回调后，上层应用需要定时调用pushExternalVideoFrame接口推送外部视频数据，每1/帧率调用一次。

   ```
   //初始化一个HWRtcVideoFrame对象，参数以实际为主，传rtcEngine
   HWRtcVideoFrame *rtcVideoFrame = [[HWRtcVideoFrame alloc] init];
    rtcVideoFrame.format =HWRtcVideoImageFormatYUV420P;
    rtcVideoFrame.width = width;
    rtcVideoFrame.height = height;
    rtcVideoFrame.data = data;
    rtcVideoFrame.dataLen = data.length;
    [rtcEngine pushExternalVideoFrame:rtcVideoFrame];
   ```

API 参考

setExternalVideoCapture

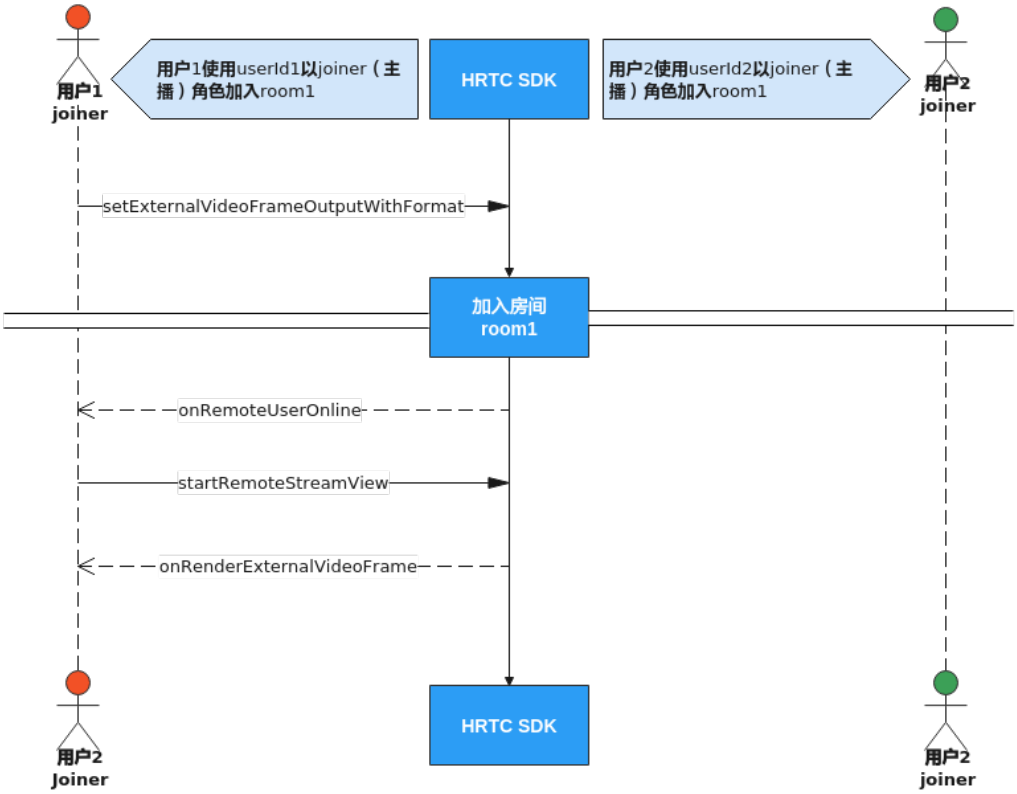
pushExternalVideoFrame

2.3.10 自定义视频渲染

功能描述

实时音视频传输过程中，上层应用可以不用SDK默认的渲染功能，选择对视频帧数据进行自定义渲染。

接口调用流程



实现过程

1. **加入房间前开启视频自渲染**

   加入房间前调用接口setExternalVideoFrameOutputWithFormat 打开视频自渲染功能。

   ```
   //开启远端或者本地的视频自渲染
   HwRtcImageBufferFormat * fileFormat = [[HwRtcImageBufferFormat alloc] init];
    fileFormat.format = HWRtcVideoImageFormatYUV420P;
    fileFormat.bufferType = HWRtcVideoImageBufferByteArray;
   [rtcEngine setExternalVideoFrameOutputWithFormat:fileFormat remoteEnable:YES localEnable:YES];
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **渲染远端视频流**

   当远端用户加入房间后会触发onRemoteUserOnline回调，在该回调中使用startRemoteStreamView方法开启接收远端用户的视频流，如果不开启，无法渲染远端视频流。

   ```
   //远端用户加入房间后触发的回调
   - (void)onRemoteUserOnline:(NSString * _Nonnull)roomId
   ```

```
            userId:(NSString * _Nonnull)userId
          userName:(NSString * _Nonnull)userName
{
//创建视频画布对象
HWRtcVideoCanvas *canvas = [[HWRtcVideoCanvas alloc] init];
canvas.uid = @”remote userId”;
canvas.view = [[HWRtcView alloc] initWitFrame:(0,0,100,100)];

// 开启远端视频流接收
 // @param remote 参考 HWRtcVideoCanvas
 // @param streamType  视频分辨率
 / /@param disableAdjustRes: 禁用分辨率自适应，默认关闭。
[rtcEngine startRemoteStreamView:canvas streamType:HWRtcStreamTypeHD disableAdjustRes:YES];
}
```

4. **触发回调**

   加入房间后sdk会根据前面的参数设置调用onRenderExternalVideoFrame回调函数上报本地和远端视频帧数据给上层应用处理。

```
//自渲染回调
- (void)onRenderExternalVideoFrame:(NSString *_Nonnull)roomid
            meidaDirection:(HWRtcMediaDirection)meidaDirection
               videoFrame:(HWRtcVideoFrame * _Nonnull)videoFrame
{
//获取videoFrame，进行数据处理
}
```

API 参考

setExternalVideoFrameOutputWithFormat

onRemoteUserOnline

startRemoteStreamView

onRenderExternalVideoFrame

2.3.11 加入多频道（跨房）

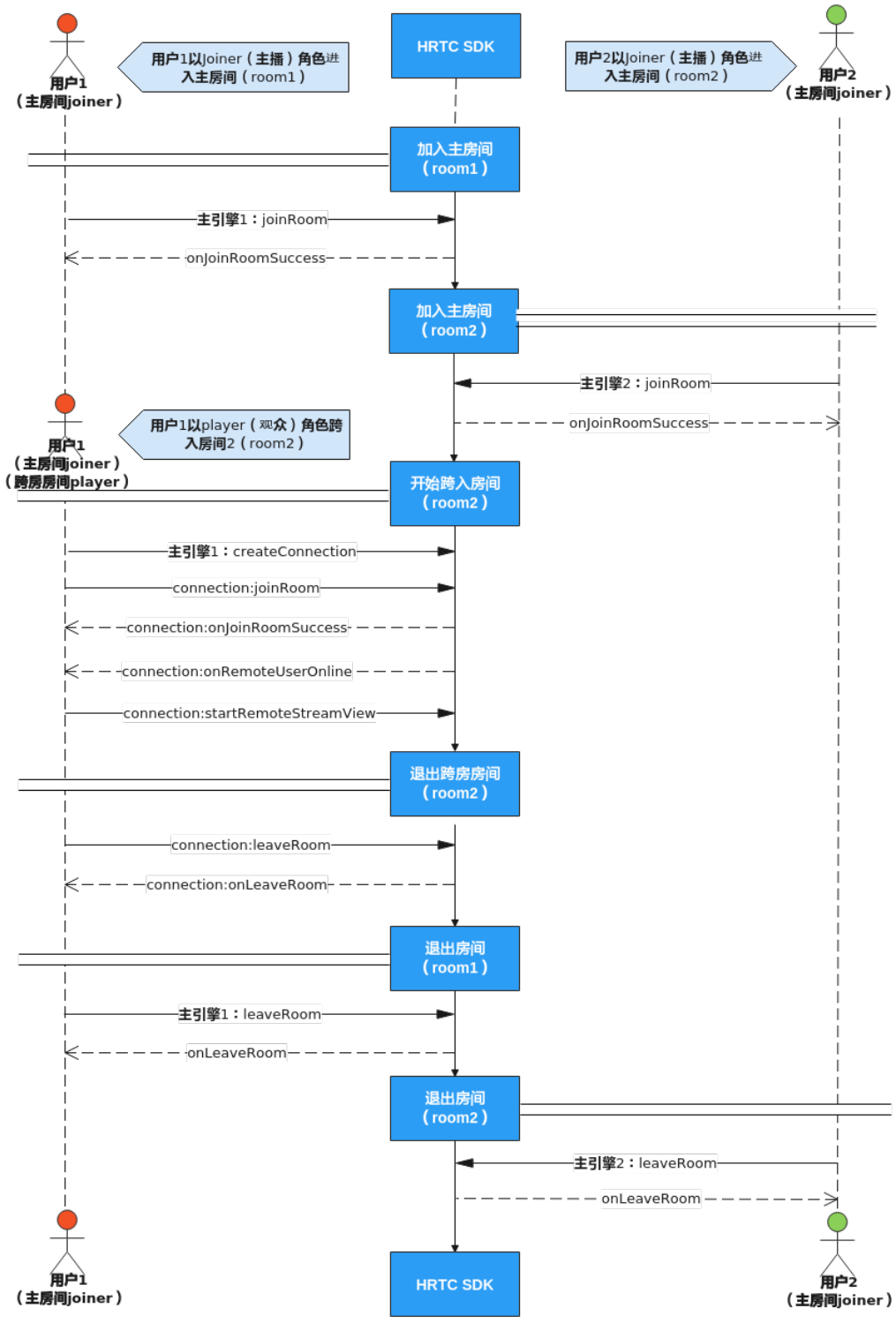
功能描述

跨房间连麦，指主播的媒体流可以同时转发进多个（目前最多支持四个）房间频道，实现主播跨频道与其他主播实时互动的场景。房间中的所有主播可以看见彼此，房间中的观众可以看到所有主播。

注意事项

- 同一时间最多只能创建4个连接对象，每个连接对象对应的房间ID必须互不相同。
- 如果使用connection对象加入房间，则加入房间的房间ID不能和已创建连接对象对应的房间ID相同。
- 同一时间只能以一个JOINER角色加入某一个房间。如果本端在其他房间里的角色是HWRtcRoleJoiner，则需要调用该房间的setUserRole方法将本端在该房间的角色切换为HWRtcRolePlayer后才能以HWRtcRoleJoiner跨入其他房间。如果本端用户是以HWRtcRolePlayer角色加入此跨房房间，则需要调用跨房连接的setUserRole方法将本端的角色切换为HWRtcRoleJoiner后才能发送音频流和视频流。

接口调用流程



实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入主房间。

2. **创建跨房连接**

当收到远端用户加入房间后会触发connection:onRemoteUserOnline跨房回调，在该回调内用对应远端用户的跨房连接调用startRemoteStreamView方法设置远端窗口并开启收流。

示例代码如下：

```
//创建跨房引擎，room id具有唯一性，不可重复，设置引擎代理
HWRtcConnection *connection = [rtcEngine createConnection:@”room id”];
connection.delegate = self;
```

3. **调用跨房连接的joinRoom加入跨房房间**

调用跨房连接的joinRoom接口加入房间，其中HWRtcUserInfo的signature鉴权签名字符串需要填入正确的计算值，计算方法请参考**接入鉴权**。

如果本端需要发送音频或者视频流给跨房房间里的远端用户，则用户的role参数需要设为HWRtcRoleJoiner，否则可以设为HWRtcRolePlayer，只收流不发流。

注意：不能同时以**JOINER**角色加入多个房间。

加入跨房房间成功后会收到跨房回调指针的onJoinRoomSuccess通知，加入房间失败会收到跨房回调指针的onJoinRoomFailure通知。

示例代码如下：

```
//创建HWRtcJoinParam对象
HWRtcJoinParam  *joinRoomParam = [[HWRtcJoinParam alloc] init];
joinRoomParam.autoSubscribeAudio = YES;
joinRoomParam.autoSubscribeVideo = YES;
joinRoomParam.userId = "userId";
joinRoomParam.userName = "userName";
joinRoomParam.authorization = authorization; //authorization：鉴权信息，具体生成方法请参见接入鉴权。与加入房间的authorization相同
joinRoomParam.ctime = time;//与加入房间的time相同
joinRoomParam.roomId = "roomId";
joinRoomParam.userRole = HWRtcRolePlayer;
joinRoomParam.sfuType = 0;
joinRoomParam.scenario = 0;
//通过跨房引擎进入房间
[connection joinRoom:joinRoomParam];
```

4. **收远端用户的视频流**

收到远端用户加入房间后会收到跨房回调connection:onRemoteUserOnline，可以调用该远端用户对应的跨房连接的startRemoteStreamView设置远端窗口并开启收流。

示例代码如下：

```
//远端用户加入房间后会收到跨房回调
- (void)connection:(HWRtcConnection *)connection
     onRemoteUserOnline:(NSString * _Nonnull)userId
          userName:(NSString * _Nonnull)userName
{
    //创建视频画布对象
    HWRtcVideoCanvas *canvas = [[HWRtcVideoCanvas alloc] init];
    canvas.uid = @”remote userId”;
    canvas.view = [[HWRtcView alloc] initWitFrame:(0,0,100,100)];
    //设置远端窗口并开启收流
    [connection startRemoteStreamView:canvas streamType:HWRtcStreamTypeHD
disableAdjustRes:YES];
}
```

5. **退出跨房房间**

跨房结束后，调用跨房连接的leaveRoom接口跨房房间。

示例代码如下：

```
//退出跨房房间
[connection leaveRoom];
```

API 参考

createConnection

joinRoom

connection:onJoinSuccess

connection:onJoinRoomFailure

connection:onRemoteUserOnline

startRemoteStreamView

leaveRoom

2.4 实现音视频通话（Windows）

2.4.1 环境准备

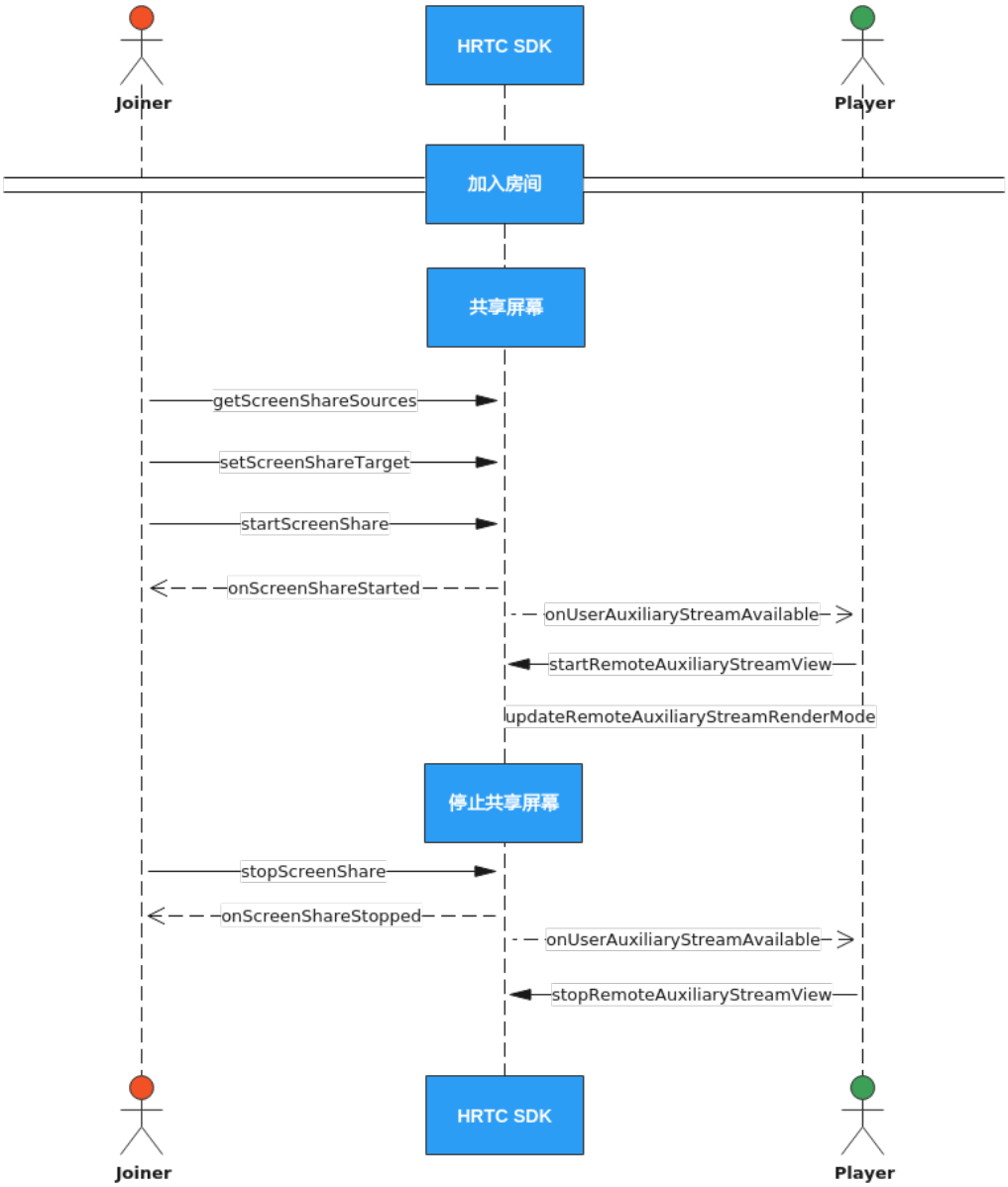
详情请参考**开发前准备**。

2.4.2 屏幕分享

功能描述

屏幕共享用于在音视频会议中，把一个与会者的屏幕内容，以视频的方式分享给其他与会者。屏幕共享可以共享整个桌面，也可以共享某一个程序窗口。共享程序窗口的时候还可以指定共享该程序窗口的矩形区域。

接口调用流程



实现屏幕共享

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **共享桌面**

   加入房间后可以调用getScreenShareSources获取屏幕共享对象列表，然后遍历找到其中HRTCScreenShareSourceInfo.type为HRTC_SCREEN_SHARE的对象，该对象就是屏幕对象，再调用setScreenShareTarget设置共享桌面参数。桌面共享不支持区域共享。

   然后调用startScreenShare开始共享。

   还可以调用setAuxiliaryVideoEncodeSmooth来切换屏幕共享的分辨率，从默认的1080p切换为720p。

   示例代码如下：

```
//获取引擎
IHRTCEngine* engine = CRtcEngineMgr::GetInstance()->GetEngine();
// 定义屏幕共享区域参数
HRTCSrceenCaptureOptionalInfo optionInfo;
memset(&optionInfo, 0, sizeof(HRTCSrceenCaptureOptionalInfo));
// 获取屏幕共享列表，包含桌面共享对象
IHRTCScreenShareSourceList* sourceList = engine-
>getScreenShareSources(HRTC_SCREENSHARE_SMALL_ICON);
unsigned int count = sourceList->count();
for (unsigned int index = 0; index < count; index++) {
    HRTCScreenShareSourceInfo sourceInfo = sourceList->get(index);
    if (sourceInfo.type == HRTC_SCREEN_SHARE) { //如果是桌面共享对象
        // 调用设置屏幕共享接口
        engine->setScreenShareTarget(&sourceInfo, &optionInfo);
        // 开启屏幕共享
        engine->startScreenShare();
        // 设置屏幕共享编码流畅优先
        engine->setAuxiliaryVideoEncodeSmooth(true);
        break;
    }
}
```

3. **共享程序**

   加入房间后也可以调用getScreenShareSources来获取可以共享的程序窗口列表，然后展示在界面上供用户选择。

   选择好需要共享的程序窗口后，调用setScreenShareTarget来设置要共享的程序窗口，其中HRTCScreenShareSourceInfo.type设为HRTC_WINDOW_CAPTURE，HRTCSrceenCaptureOptionalInfo参数用于指定共享的程序窗口的子矩形区域，矩形区域以共享的程序窗口的左上角为原点(0,0)，最小96*92 最大1920*1080。如果需要共享该程序的完整窗口，则**HRTCSrceenCaptureOptionalInfo**参数的矩形区域全填为0。

   然后调用startScreenShare开始共享。

   当HRTCSrceenCaptureOptionalInfo参数的矩形区域填全0时，还可以调用setAuxiliaryVideoEncodeSmooth来切换屏幕共享的分辨率，从默认的1080p切换为720p。

   示例代码如下：

```
//获取共享程序窗口列表
IHRTCEngine* engine = CRtcEngineMgr::GetInstance()->GetEngine();
IHRTCScreenShareSourceList* sourceList = engine-
>getScreenShareSources(HRTC_SCREENSHARE_SMALL_ICON);
unsigned int count = sourceList->getCount();
for (unsigned int index = 0; index < count; index++) {
HRTCScreenShareSourceInfo sourceInfo = sourceList->get(index);
RD_LOG("appshare", "applist name: %s, HWND:%x", sourceInfo.sourceName, sourceInfo.sourceId);
//将sourceInfo加入列表展示到界面上供用户选择
}

//用户选择共享的程序后
int nIdx = m_appList.GetSelectionMark(); //获取用户选择的程序窗口序号
//获取用户选择的程序窗口序号对应的窗口信息对象
HRTCScreenShareSourceInfo sourceInfo = m_windowsList[nIdx];
//获取之前创建的主引擎指针
IHRTCEngine* engine = CRtcEngineMgr::GetInstance()->GetEngine();
//设置屏幕共享参数
HRTCSrceenCaptureOptionalInfo optionInfo;
memset(&optionInfo, 0, sizeof(HRTCSrceenCaptureOptionalInfo));
//根据需要设置共享的程序窗口的子矩形区域，如果要共享完整的程序窗口，则矩形区域设为全0
optionInfo.rect.left = 0;
optionInfo.rect.top = 0;
optionInfo.rect.right = 320;
optionInfo.rect.bottom = 240;
int ret = engine->setScreenShareTarget(&sourceInfo, &optionInfo);
ret = engine->startScreenShare();
```

4. **接收远端用户的共享流**

   收到远端用户开启共享流通知onUserAuxiliaryStreamAvailable后，可以调用startRemoteAuxiliaryStreamView来设置远端用户的共享流的窗口句柄并开始选看。

   还可以再调用updateRemoteRenderMode设置窗口显示共享流的方式。

   示例代码如下：

   ```
   engine->startRemoteAuxiliaryStreamView(userId, wndHandle);
   ```

5. **停止屏幕共享**

   屏幕共享结束时，可以调用stopScreenShare停止屏幕共享。

   示例代码如下：

   ```
   engine->stopScreenShare();
   ```

6. **停止接收远端用户的屏幕共享流**

   收到onUserAuxiliaryStreamAvailable消息后，如果选看的远端屏幕共享流不可用，或者收到远端用户下线通知onRemoteUserOffline，则接收端必须调用stopRemoteAuxiliaryStreamView关闭共享流窗口视图。

   如果接收端需要主动停止接收远端用户的共享流，也需要调用stopRemoteAuxiliaryStreamView接口停止接收共享流。

   示例代码如下：

   ```
   engine->stopRemoteAuxiliaryStreamView(userId);
   ```

API 参考

getScreenShareSources

setScreenShareTarget

setAuxiliaryVideoEncodeSmooth

startScreenShare

stopScreenShare

startRemoteAuxiliaryStreamView

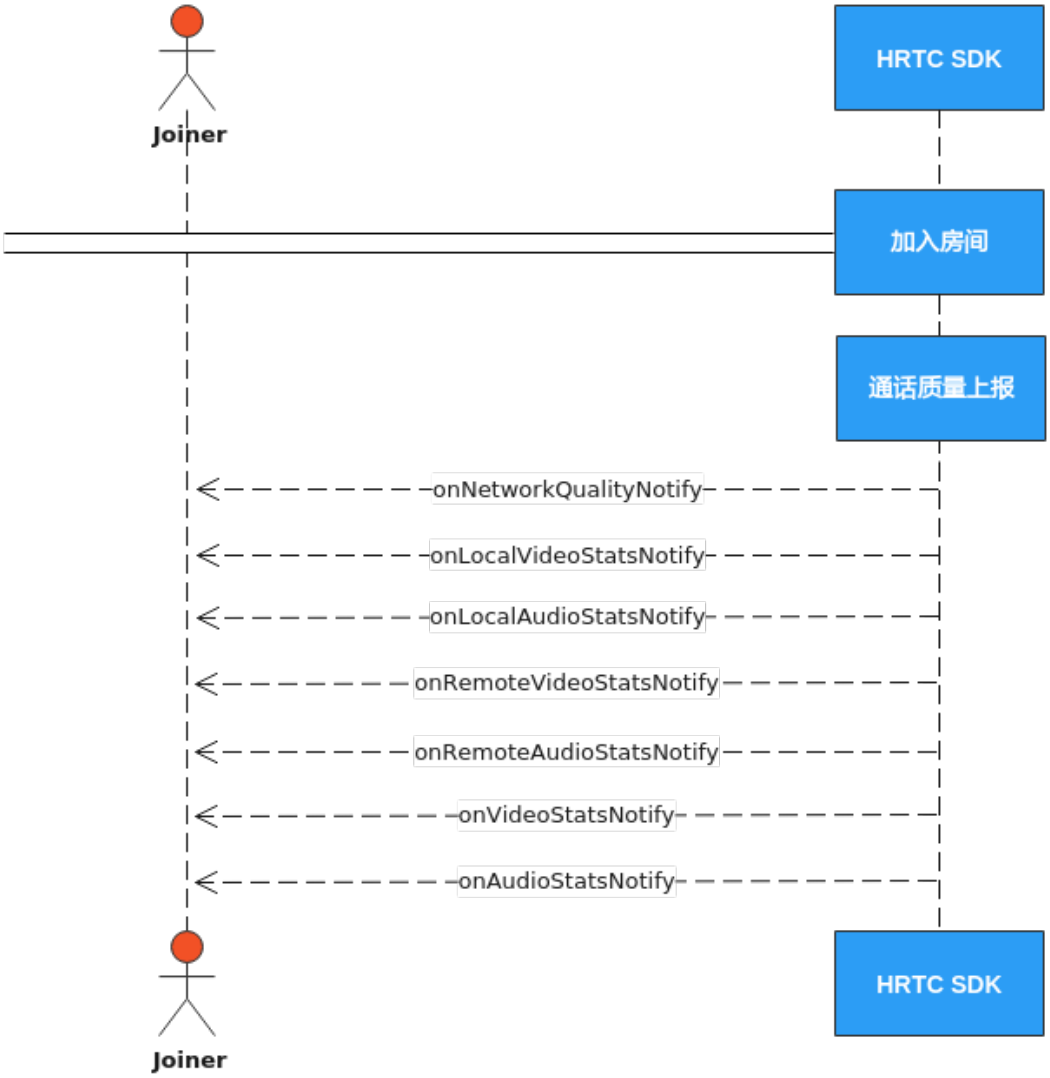
stopRemoteAuxiliaryStreamView

2.4.3 通话质量监测

功能描述

加入频道后，SDK会每隔2秒自动触发通话质量相关的回调，上报当前通话的网络质量、本地和远端的音视频统计信息。

上报接口



通话质量上报

onNetworkQualityNotify回调上报当前通话中每个入会者的上下行网络质量。默认开启，每2s上报一次。

```
void HWEngineEventHandler::onNetworkQualityNotify(HRTCQualityInfo * localQuality, unsigned int localQualityCount, HRTCQualityInfo * remoteQuality, unsigned int remoteQualityCount) {
    for (unsigned int i = 0; i < remoteQualityCount; i++) {
        HRTCQualityInfo stats;
        memcpy_s(&stats, sizeof(stats), &remoteQuality[i], sizeof(HRTCQualityInfo));
        //把远端网络质量刷新到统计界面上
    }
}
```

本地音频流统计信息报告

onLocalAudioStatsNotify回调上报本地设备发送音频流的统计信息。您可以了解到当前通话声道数（单声道或双声道）、发送音频的采样率、码率、比特率、丢包率、延时和抖动等。

```
void HWEngineEventHandler::onLocalAudioStatsNotify(const HRTCLocalAudioStats * localStats, unsigned int
localStatsCount) {
}
```

远端音频流统计信息报告

onRemoteAudioStatsNotify回调上报当前通话中每个远端用户音频流的统计信息。您可以了解到每个远端用户发送的音频流的采样率、声道数、码率、丢包率、延时、抖动和卡顿时长等一些信息。

```
void HWEngineEventHandler::onRemoteAudioStatsNotify(const HRTCRemoteAudioStats * remoteStats,
unsigned int remoteStatsCount) {
}
```

本地视频流统计信息上报

onLocalVideoStatsNotify回调向您报告当前本地视频流统计信息，包括帧率、码率、延时、抖动等信息。

```
void HWEngineEventHandler::onLocalVideoStatsNotify(const HRTCLocalVideoStats * localStats, unsigned int
localStatsCount) {
}
```

远端视频流状态监控

onRemoteVideoStatsNotify回调向您报告当前远端视频流统计信息，包括帧率、码率、延时、抖动和卡顿时长等信息。

```
void HWEngineEventHandler::onRemoteVideoStatsNotify(const HRTCRemoteVideoStats * remoteStats,
unsigned int remoteStatsCount) {
}
```

视频流状态监控

onVideoStatsNotify回调上报视频流的状态，包括本地上行视频流和远端用户的下行视频流状态。

```
void HWEngineEventHandler::onVideoStatsNotify(HRTCLocalVideoStats *localStats, unsigned int
localStatsCount, HRTCRemoteVideoStats *remoteStats, unsigned int remoteStatsCount) {
    for (unsigned int i = 0; i < localStatsCount; i++) {
        HRTCLocalVideoStats stats;
        memcpy_s(&stats, sizeof(stats), &localStats[i], sizeof(HRTCLocalVideoStats));
        //本地视频流信息刷新到统计界面上
    }
    for (unsigned int i = 0; i < remoteStatsCount; i++) {
        HRTCRemoteVideoStats stats;
        memcpy_s(&stats, sizeof(stats), &remoteStats[i], sizeof(HRTCRemoteVideoStats));
        //远端视频流信息刷新到统计界面上
    }
}
```

音频流状态监控

onAudioStatsNotify回调上报音频流的状态，包括本地上行音频流和远端用户的下行音频流状态。

```
void HWEngineEventHandler::onAudioStatsNotify(HRTCLocalAudioStats *localStats, unsigned int
localStatsCount, HRTCRemoteAudioStats *remoteStats, unsigned int remoteStatsCount) {
    for (unsigned int i = 0; i < localStatsCount; i++) {
        HRTCLocalAudioStats stats;
        memcpy_s(&stats, sizeof(stats), &localStats[i], sizeof(HRTCLocalAudioStats));
        //本地音频流信息刷新到统计界面上
```

```
    }
    for (unsigned int i = 0; i < remoteStatsCount; i++) {
        HRTCRemoteAudioStats stats;
        memcpy_s(&stats, sizeof(stats), &remoteStats[i], sizeof(HRTCRemoteAudioStats));
        //远端音频流信息刷新到统计界面上
    }
}
```

API 参考

onNetworkQualityNotify

onLocalVideoStatsNotify

onLocalAudioStatsNotify

onRemoteVideoStatsNotify

onRemoteAudioStatsNotify

onVideoStatsNotify

onAudioStatsNotify

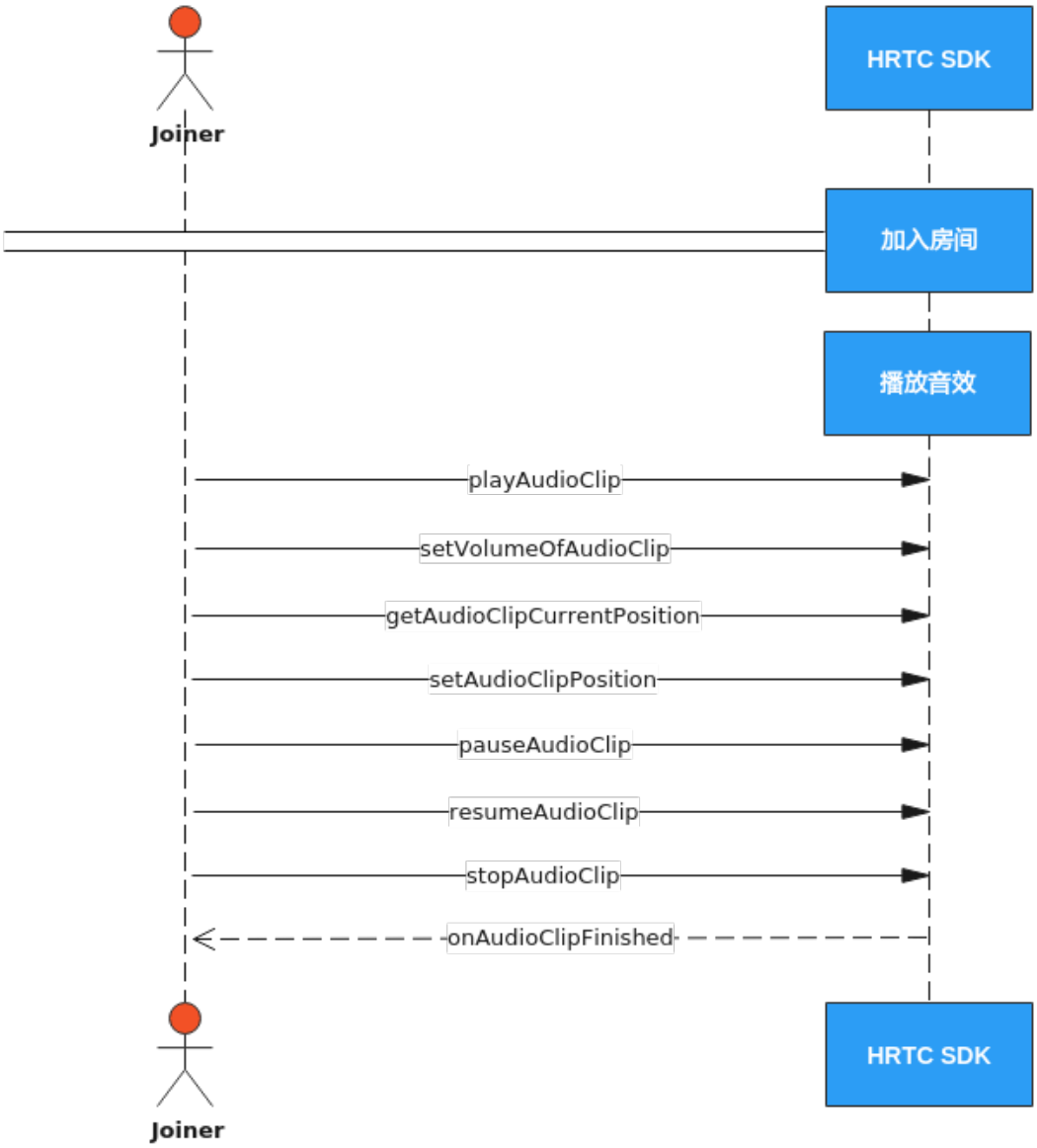
2.4.4 播放音效

功能描述

用户可以同时播放多个音效文件，给自己和其他与会者听，用于烘托气氛。

支持本地或在线文件路径，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程



实现流程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **播放音效文件**

   调用playAudioClip接口可以播放一个音效文件。可以同时播放多个音效文件，不同音效文件用不同的soundId参数进行区别。

   ```
   int soundId = 0; //音效文件ID
   engine->playAudioClip(soundId, "D:\\test.mp3", 1, 0, 0, 100, true, 0);
   ```

3. **设置音效文件音量**

   音效文件播放过程中，可以调用setVolumeOfAudioClip设置音效文件播放音量。

   音量大小，范围为0-100。

   ```
   int soundId = 0; //音效文件ID
   engine->setVolumeOfAudioClip(soundId, 50);
   ```

4. **获取音效文件总时长**

   播放过程中可以调用getAudioClipDuration获取音效文件总时长，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int duration = engine->getAudioClipDuration(soundId, "D:\\test.mp3");
   //根据当前总时长刷新界面进度条
   ```

5. **获取音效文件播放位置**

   播放过程中可以调用getAudioClipCurrentPosition获取音效文件播放位置，可用于刷新界面的播放进度条。

   ```
   int soundId = 0; //音效文件ID
   int pos = engine->getAudioClipCurrentPosition(soundId);
   //根据当前播放位置刷新界面进度条
   ```

6. **设置音效文件播放位置**

   播放过程中可以调用setAudioClipPosition设置音效文件播放位置，可用于通过拖动进度条改变音效文件的播放位置。

   ```
   int soundId = 0; //音效文件ID
   engine->setAudioClipPosition(soundId, 50);
   ```

7. **暂停播放音效文件**

   调用pauseAudioClip接口可以暂停播放一个音效文件。调用pauseAllAudioClips暂停播放所有正在播放的音效文件。

   ```
   int soundId = 0; //音效文件ID
   engine->pauseAudioClip(soundId);
   ```

8. **恢复播放暂停的音效文件**

   音频文件暂停播放后，可以调用resumeAudioClip接口可以恢复播放之前暂停的音效文件。或者调用resumeAllAudioClips恢复播放所有暂停的音效文件。

   ```
   int soundId = 0; //音效文件ID
   engine->resumeAudioClip(soundId);
   ```

9. **停止播放音效文件**

   调用stopAudioClip接口停止播放一个音效文件，或者调用stopAllAudioClips接口停止播放所有音效文件。

   ```
   int soundId = 0; //音效文件ID
   engine->stopAudioClip(soundId);
   ```

10. **音效文件播放结束回调**

    音效文件播放结束后，sdk会触发onAudioClipFinished回调来通知上层应用。

    ```
    void HWEngineEventHandler::onAudioClipFinished(int soundId)
    {
    //收到音效文件播放结束通知后，可以刷新界面，比如恢复播放前初始界面状态。
    }
    ```

API 参考

playAudioClip

setVolumeOfAudioClip

getAudioClipDuration

getAudioClipCurrentPosition

setAudioClipPosition

pauseAudioClip

pauseAllAudioClips

resumeAudioClip

resumeAllAudioClips

stopAudioClip

stopAllAudioClips

onAudioClipFinished

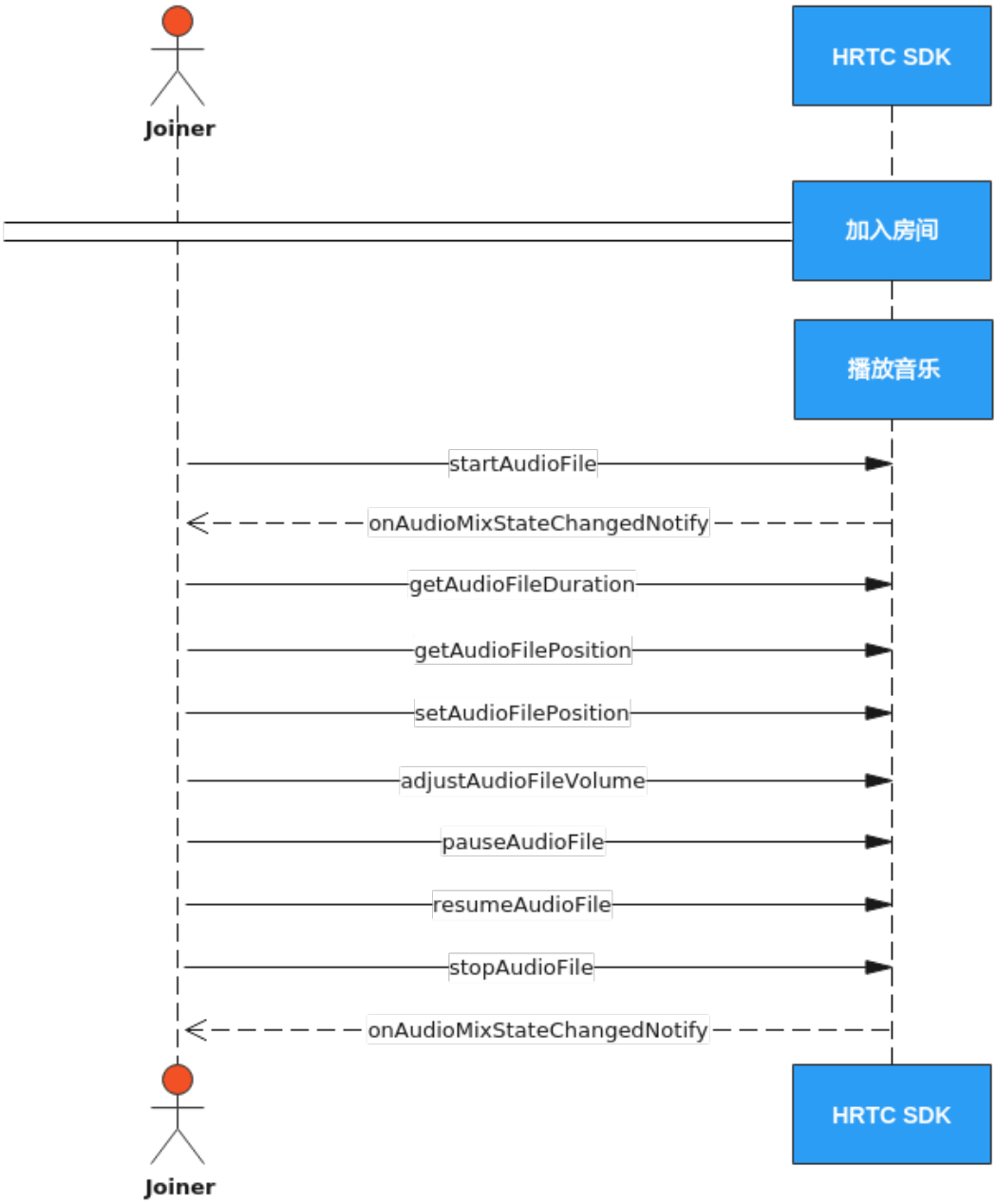
2.4.5 播放音乐

功能描述

混音是将音乐文件和麦克风音频混合，一般用于播放时长较长的背景音乐或者伴奏。同一时间只能播放一个音乐文件。可以在本地播放，也可以同时播放给其他与会者听。

支持播放wav、pcm和单声道mp3音频文件。可以播放本地或在线音乐文件。

接口调用步骤



实现过程

1. **加入房间**

   参考**接口调用步骤**中加入房间的时序图步骤加入房间。

2. **播放音乐文件**

   调用startAudioFile接口可以播放一个音乐文件。同一时刻只能播放一个音乐文件。

   ```
   engine->startAudioFile("D:\\test.mp3", true, 1, false, 0);
   ```

3. **设置音乐文件音量**

   音乐文件播放过程中，可以调用adjustAudioFileVolume设置音乐文件播放音量。

   ```
   engine->adjustAudioFileVolume(50);
   ```

4. **获取音乐文件总时长和播放位置**

   音乐文件打开成功后，Sdk会触发一次onAudioMixStateChangedNotify回调，传入state参数为HRTC_AUDIO_FILE_OPEN_COMPLETED，此时可以调用getAudioFileDuration获取音乐文件总时长，可用以刷新界面进度条的总时长。

   音乐文件在播放过程中，Sdk每秒会触发一次onAudioMixStateChangedNotify回调，其中的state参数为HRTC_AUDIO_FILE_POSITION_UPDATE，value参数就是当前的音乐文件播放进度，以毫秒为单位，可以在此回调中刷新界面的播放进度条。

   播放过程中也可以调用getAudioFilePosition获取音乐文件当前播放位置，可用于刷新界面的播放进度条。示例代码如下：

```
void HWEngineEventHandler::onAudioMixStateChangedNotify(HRTCAudioFileState state,
HRTCAudioFileReason reason, unsigned long long value)
{
switch (reason) {
case HRTC_AUDIO_FILE_REASON_NONE:
{
switch (state) {
case HRTC_AUDIO_FILE_OPEN_COMPLETED:
{
//获取总时长，并刷新到进度条
int mixDuration = engine->getAudioFileDuration();
break;

}
case HRTC_AUDIO_FILE_POSITION_UPDATE:
{
//使用value参数值刷新界面的播放进度条
break;
}
case HRTC_AUDIO_FILE_PLAYING:
{
//刷新界面
break;
}
case HRTC_AUDIO_FILE_PAUSED:
{
//刷新界面
break;
}
case HRTC_AUDIO_FILE_STOPPED:
{
//刷新界面
break;
}
case HRTC_AUDIO_FILE_PLAY_COMPLETED:
{
//刷新界面
break;
}
}
break;
}
Default:
//错误信息提示
}
}
```

5. **设置音乐文件播放位置**

   播放过程中可以调用setAudioFilePosition设置音乐文件播放位置，可用于通过拖动进度条改变音乐文件的播放位置。

```
engine-> setAudioFilePosition(50);
```

6. **暂停播放音乐文件**

   调用pauseAudioFile接口可以暂停播放一个音乐文件。

```
engine->pauseAudioFile();
```

7. **恢复播放暂停的音乐文件**

   音乐文件暂停播放后，可以调用resumeAudioFile接口恢复播放。

```
engine->resumeAudioFile();
```

8. **停止播放音乐文件**

   调用stopAudioFile接口可以停止播放音乐文件。

```
engine->stopAudioFile();
```

9. **音乐文件播放结束回调**

   音乐文件播放结束后，sdk会触发onAudioMixStateChangedNotify回调，传入state参数为HRTC_AUDIO_FILE_PLAY_COMPLETED来通知上层应用。

```
void HWEngineEventHandler::onAudioMixStateChangedNotify(HRTCAudioFileState state,
HRTCAudioFileReason reason, unsigned long long value)
{
    //收到音乐文件播放结束通知后，可以刷新界面，比如恢复播放前初始界面状态。
    if (state == HRTC_AUDIO_FILE_PLAY_COMPLETED) {
    }
}
```

API 参考

startAudioFile

adjustAudioFileVolume

getAudioFileDuration

getAudioFilePosition

setAudioFilePosition

pauseAudioFile

resumeAudioFile

stopAudioFile

onAudioMixStateChangedNotify

2.4.6 原始音频数据（音频前后处理）

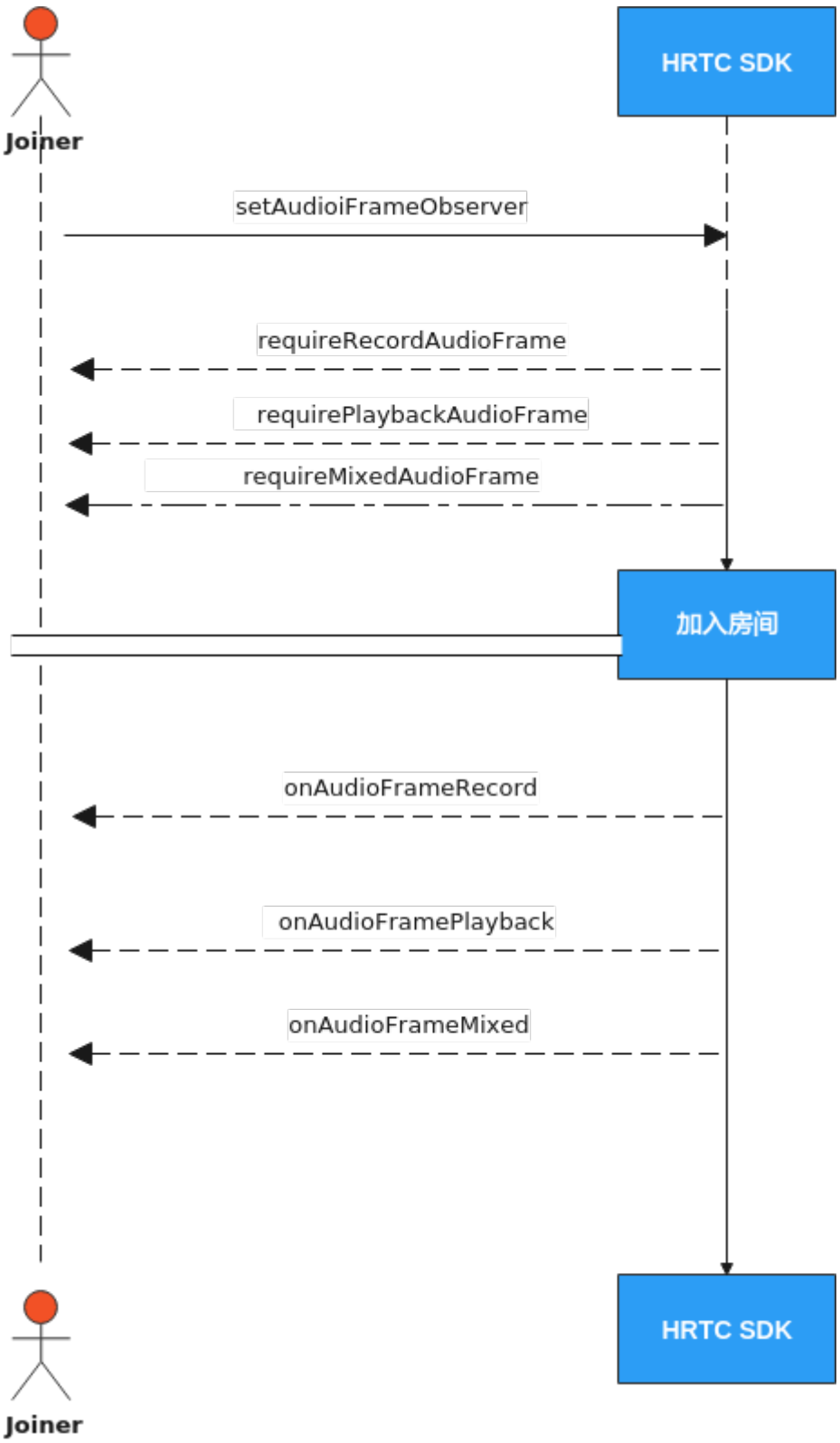
功能描述

音视频传输过程中，我们可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

对于有自行处理音视频数据需求的场景，HWRtcEngine SDK 提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行音频前处理，然后发送给远端。也可以进行音频后处理。

接口调用流程



注册音频前后处理

获取HWRtcEngie的HWRtcMediaEngie对象。

```
IHRTCMediaEngine * mediaEngine = m_pRtcEngine->getMediaEngine();
```

注册音频前后处理

```
mediaEngine->setAudioFrameObserver(&g_AudioObser);
```

每次入会都需要重新注册。取消注册，则传nil。

HRTCAudioFrameObserver g_AudioObser要继承IHRTCAudioFrameObserver。

和实现

```
requireRecordAudioFrame:
requirePlaybackAudioFrame:
requireMixedAudioFrame:
```

回调，通过回调的返回值来决定对应音频帧的处理是否生效。

实现

```
onAudioFramePlayback:
onAudioFrameMixed:
onAudioFrameRecord:
```

回调，从回调中获取音频帧并进行处理。

注意：所有回调的返回值为false，说明对音频帧的处理无效。

示例代码如下：

```
class HRTCAudioFrameObserver : public IHRTCAudioFrameObserver {
public:
    HRTCAudioFrameObserver() {
        RTCReadPcmFile();
    }
    virtual bool onAudioFramePlayback(HRTCAudioFrame& auidoFrame);
    virtual bool onAudioFrameMixed(HRTCAudioFrame& audioFrame);
    virtual bool onAudioFrameRecord(HRTCAudioFrame& audioFrame);

    virtual bool requireRecordAudioFrame() override;
    virtual bool requirePlaybackAudioFrame() override;
    virtual bool requireMixedAudioFrame() override;
};
/// 音频前后处理
bool HRTCAudioFrameObserver::onAudioFramePlayback(HRTCAudioFrame& audioFrame) {
    //远端音频数据
    return true;
}
bool HRTCAudioFrameObserver::onAudioFrameMixed(HRTCAudioFrame & audioFrame) {
    //混音数据
    return true;
}
bool HRTCAudioFrameObserver::onAudioFrameRecord(HRTCAudioFrame& audioFrame) {
    //本地音频数据
    return true;
}
bool HRTCAudioFrameObserver::requireRecordAudioFrame() {
    return true;
}
bool HRTCAudioFrameObserver::requirePlaybackAudioFrame() {
    return true;
}
bool HRTCAudioFrameObserver::requireMixedAudioFrame() {
    return true;
}
HRTCAudioFrameObserver g_AudioObser;
IHRTCMediaEngine * mediaEngine = m_pRtcEngine->getMediaEngine();
mediaEngine->setAudioFrameObserver(&g_AudioObser);
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setAudioFrameObserver

requireRecordAudioFrame

requirePlaybackAudioFrame

requireMixedAudioFrame

onAudioFramePlayback

onAudioFrameRecord

onAudioFrameMixed

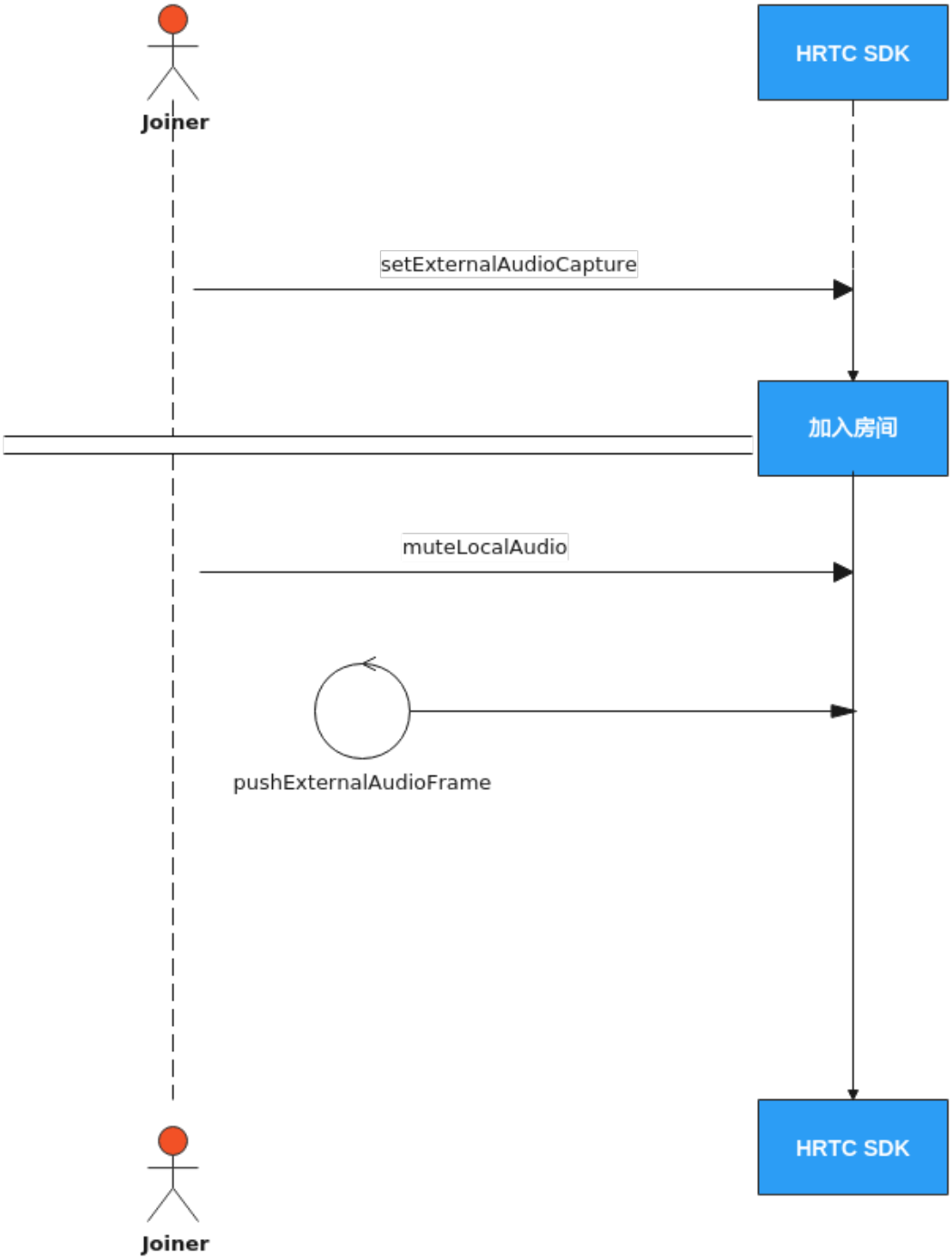
2.4.7 音频自采集和音频自渲染

功能介绍

实时音频传输过程中，HWRtcEngine SDK 通常会启动默认的音频模块进行采集和渲染。在以下场景中，您可能会发现默认的音频模块无法满足开发需求：

- app中已有自己的音频模块。
- 需要使用自定义的采集或播放处理。
- 某些音频采集设备被系统独占。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalAudioCapture（true）**

   加入房间前调用此接口打开自采集功能。

   ```
   m_pRtcEngine->setExternalAudioCapture(true, sampleRate, channels);
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **开启音频流发送**

   调用接口muteLocalAudio开启音频流发送。

   ```
   m_pRtcEngine-> muteLocalAudio(false);
   ```

4. **定时推送音频数据帧**

   加入房间成功回调后，每隔10ms定时调用pushExternalAudioFrame接口推送外部音频数据。

   音频输入数据大小：10 * sampleRate * channels * 16 / 8 / 1000字节，其中的sampleRate和channels是前面调用的setExternalAudioCapture里传入的采样率和声道数参数。

   ```
   // audioData  获取的音频数据指针， size 获取的音频数据字节数
    engine->pushExternalAudioFrame(audioData, size);
   ```

   根据帧率循环调用pushExternalAudioFrame方法往SDK推送数据。

5. **音频自渲染**

   暂不支持音频自渲染功能。

API 参考

setExternalAudioCapture

muteLocalAudio

pushExternalAudioFrame

2.4.8 原始视频数据（视频前后处理）

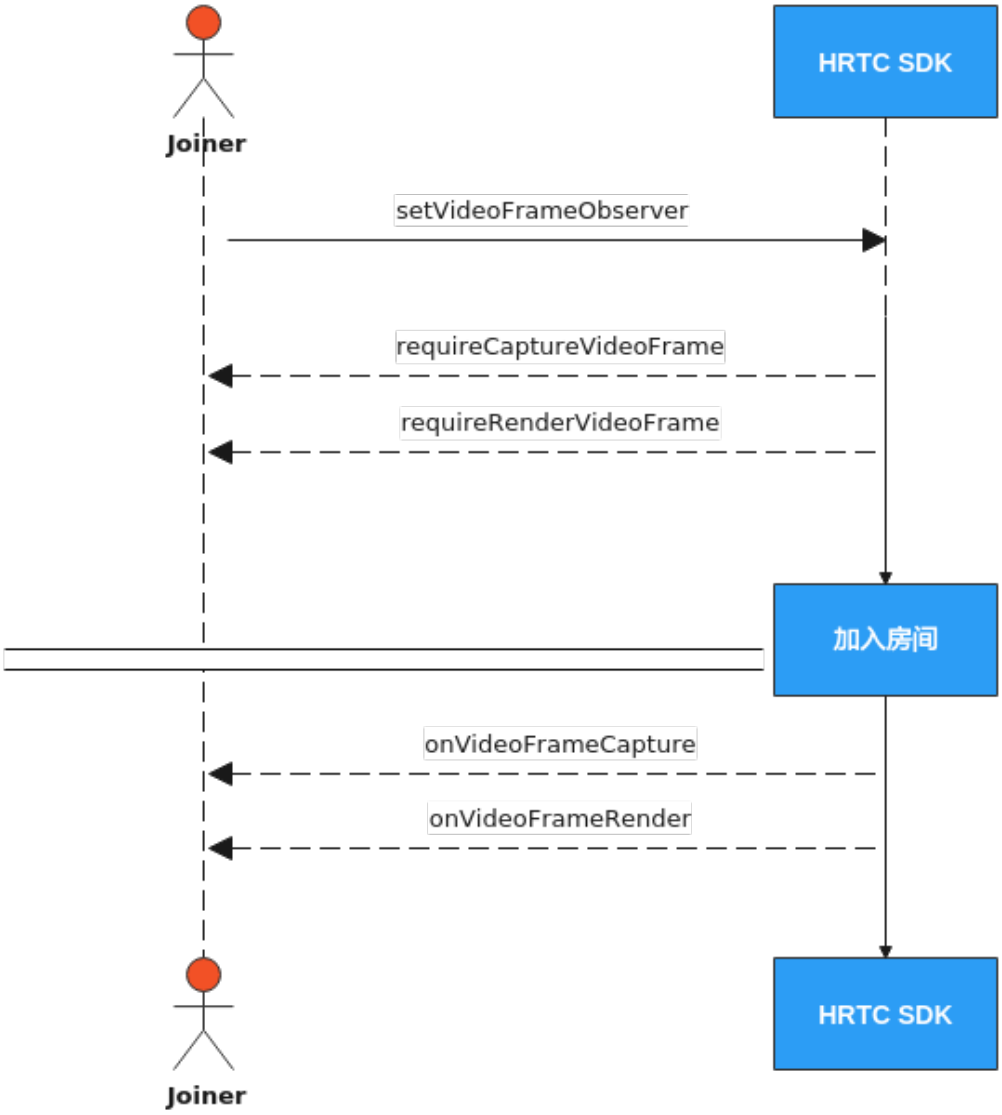
功能描述

音视频传输过程中，可以对采集到的音视频数据进行前处理和后处理，获取想要的播放效果。

对于有自行处理音视频数据需求的场景，HWRtcEngine SDK提供原始数据功能，您可以在将数据发送给编码器前进行前处理，对捕捉到的音频信号或视频帧进行修改，也可以在将数据发送给解码器后进行后处理，对接收到的音频信号或视频帧进行修改。

原始音频数据可以进行视频前处理，然后发送给远端。也可以进行视频后处理。

接口调用流程



注册视频前后处理

获取HWRtcEngie的HWRtcMediaEngie对象。

```
IHRTCMediaEngine * mediaEngine = m_pRtcEngine->getMediaEngine();
```

注册视频前后处理

```
mediaEngine->setVideoFrameObserver(&g_VideoObser);
```

每次入会都需要重新注册。取消注册，则传nil。

HRTCVideoFrameObserver g_VideoObser要继承IHRTCVideoFrameObserver

和实现

```
requireCaptureVideoFrame:
requireRenderVideoFrame:
```

回调，通过回调的返回值来决定对应视频帧的处理是否生效。

实现


```
onVideoFrameCapture：
onVideoFrameRender：
```

回调，从回调中获取视频帧并进行处理。

注意：所有回调的返回值为false，说明对视频帧的处理无效。

示例代码如下：

```
class HRTCVideoFrameObserver : public IHRTCVideoFrameObserver {
public:
    virtual bool onVideoFrameCapture(HRTCVideoFrame& videoFrame);
    virtual bool onVideoFrameRender(const char* userId, HRTCVideoFrame& videoFrame);
    virtual bool requireCaptureVideoFrame() override;
    virtual bool requireRenderVideoFrame()override;
};
//视频前处理
bool HRTCVideoFrameObserver::onVideoFrameCapture(HRTCVideoFrame& videoFrame) {
    return true;
}
//后处理
bool HRTCVideoFrameObserver::onVideoFrameRender(const char* userId, HRTCVideoFrame& videoFrame) {
    return true;
}
bool HRTCVideoFrameObserver::requireCaptureVideoFrame() {
    return true;
}
bool HRTCVideoFrameObserver::requireRenderVideoFrame() {
    return true;
}
```

加入房间

参考**接口调用流程**中加入房间的时序图步骤加入房间。

API 参考

setVideoFrameObserver

requireCaptureVideoFrame

requireRenderVideoFrame

onVideoFrameCapture

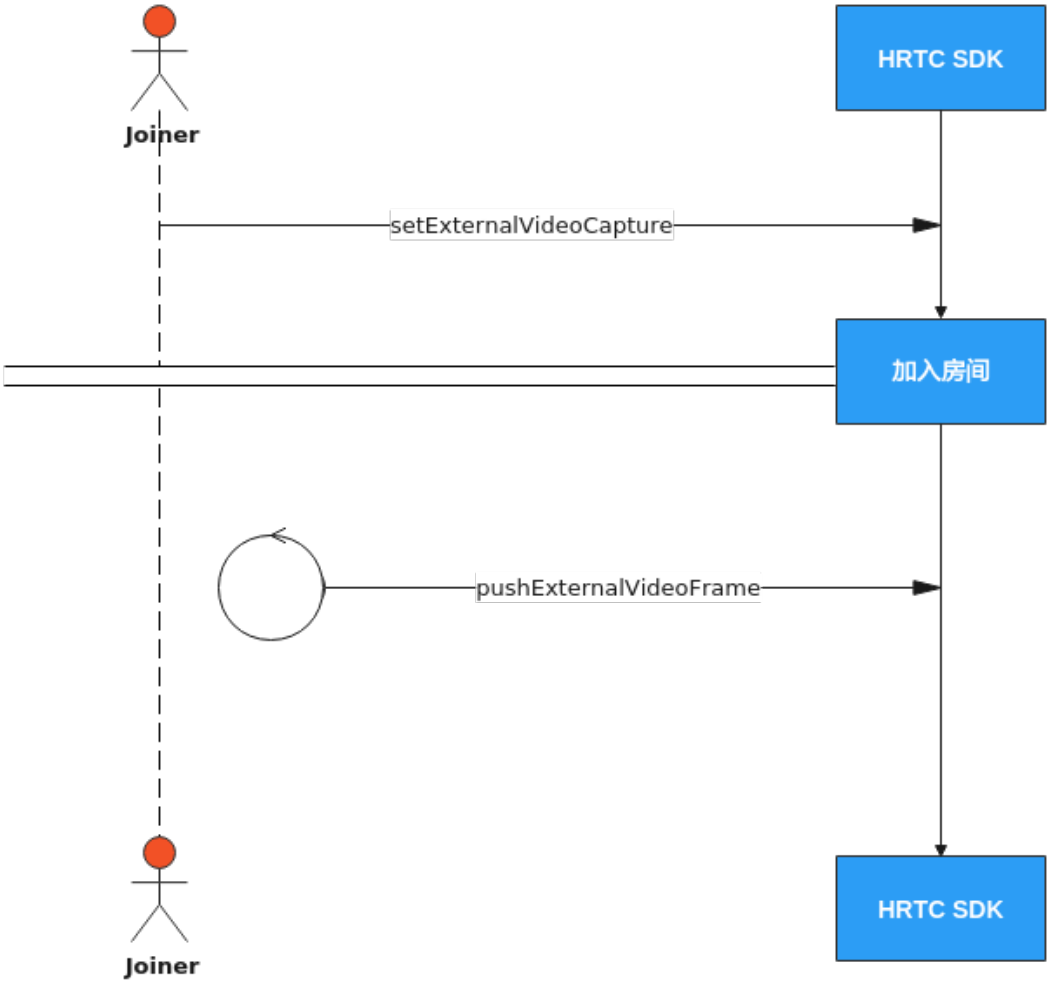
onVideoFrameRender

2.4.9 自定义视频采集

功能描述

如果您需要使用自定义的美颜库或有前处理库，则需要自己采集和处理摄像头拍摄画面，您可以通过SparkRTC SDK的setExternalVideoCapture接口开启自采集功能。然后使用pushExternalVideoFrame接口推送外部视频数据到SparkRTC SDK播放。

接口调用流程



实现过程

1. **加入房间前调用主引擎的setExternalVideoCapture**

   加入房间前调用此接口打开视频自采集功能。一旦开启后，你将无法切换。

   ```
   //开启自采集功能
   m_pRtcEngine->setExternalVideoCapture(true, HRTC_VIDEO_IMAGE_FORMAT_YUV420P);
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **定时推送视频数据帧**

   加入房间成功回调后，上层应用需要定时调用pushExternalVideoFrame接口推送外部视频数据，每1/帧率调用一次。

   ```
   //初始化一个HWRtcVideoFrame对象，参数以实际为主，传rtcEngine
   HWRtcVideoFrame *rtcVideoFrame = [[HWRtcVideoFrame alloc] init];
    rtcVideoFrame.format =HWRtcVideoImageFormatYUV420P;
    rtcVideoFrame.width = width;
    rtcVideoFrame.height = height;
    rtcVideoFrame.data = data;
    rtcVideoFrame.dataLen = data.length;
    rtcEngine->pushExternalVideoFrame(&rtcVideoFrame);
   ```

API 参考

setExternalVideoCapture

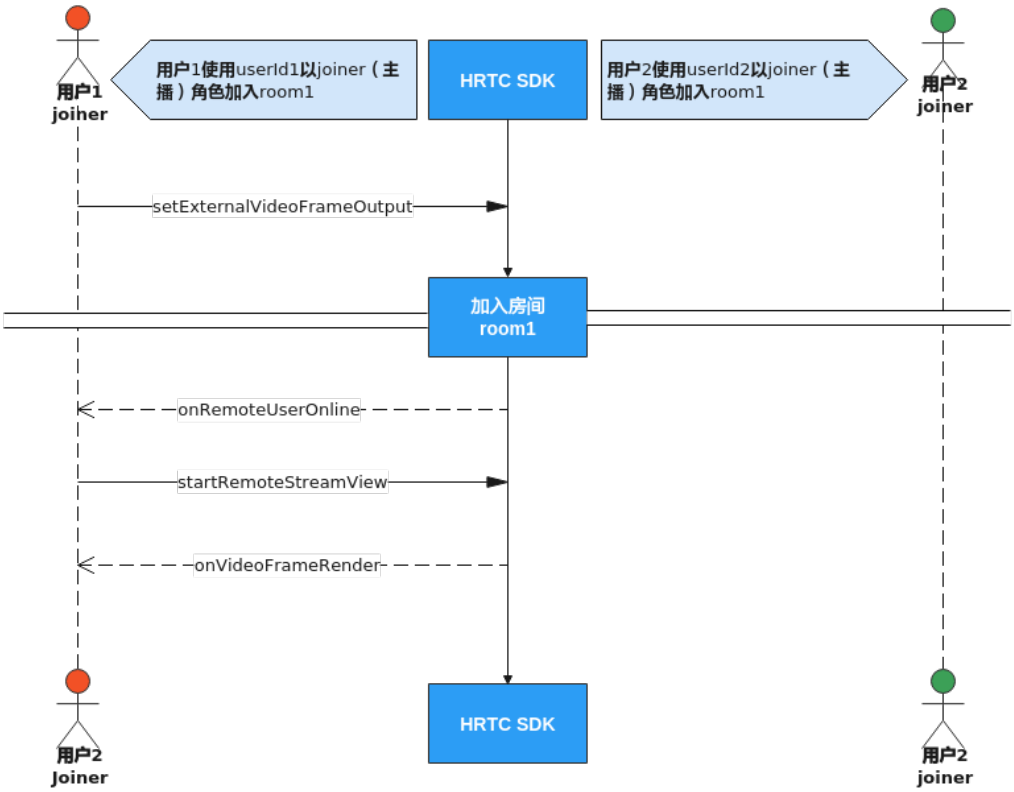
pushExternalVideoFrame

2.4.10 自定义视频渲染

功能描述

实时音视频传输过程中，上层应用可以不用SDK默认的渲染功能，选择对视频帧数据进行自定义渲染。

接口调用流程



实现过程

1. **加入房间前开启视频自渲染**

   加入房间前调用接口setExternalVideoFrameOutput打开视频自渲染功能。

   ```
   //开启远端或者本地的视频自渲染
   HRTCImageBufferFormat fileFormat;
   fileFormat.format = HRTC_VIDEO_IMAGE_FORMAT_YUV420P;
   fileFormat.bufferType = HRTC_VIDEO_IMAGE_BUFFER_BYTE_ARRAY;
   rtcEngine->setExternalVideoFrameOutput(true, true, fileFormat);
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

3. **渲染远端视频流**

当远端用户加入房间后会触发onRemoteUserOnline回调，在该回调中使用pullRemoteVideo方法开启接收远端用户的视频流，如果不开启，无法渲染远端视频流。

```
//远端用户加入房间后触发的回调
void HWEngineEventHandler::onRemoteUserOnline(const char* roomId, const char* userId, const char* userName) {
    // 开启远端视频流接收
    rtcEngine->pullRemoteVideo(userId, true);
}
```

4. **触发回调**

   加入房间后sdk会根据前面的参数设置调用onVideoFrameRender回调函数上报本地和远端视频帧数据给上层应用处理。

```
//自渲染回调
void HWEngineEventHandler::onVideoFrameRender(const char* roomId, HRTCMediaDirection direction, const char* userId, HRTCVideoFrame& videoFrame) {
    //获取videoFrame，进行数据处理
}
```

API 参考

setExternalVideoFrameOutput

onRemoteUserOnline

startRemoteStreamView

onVideoFrameRender

2.4.11 加入多频道（跨房）

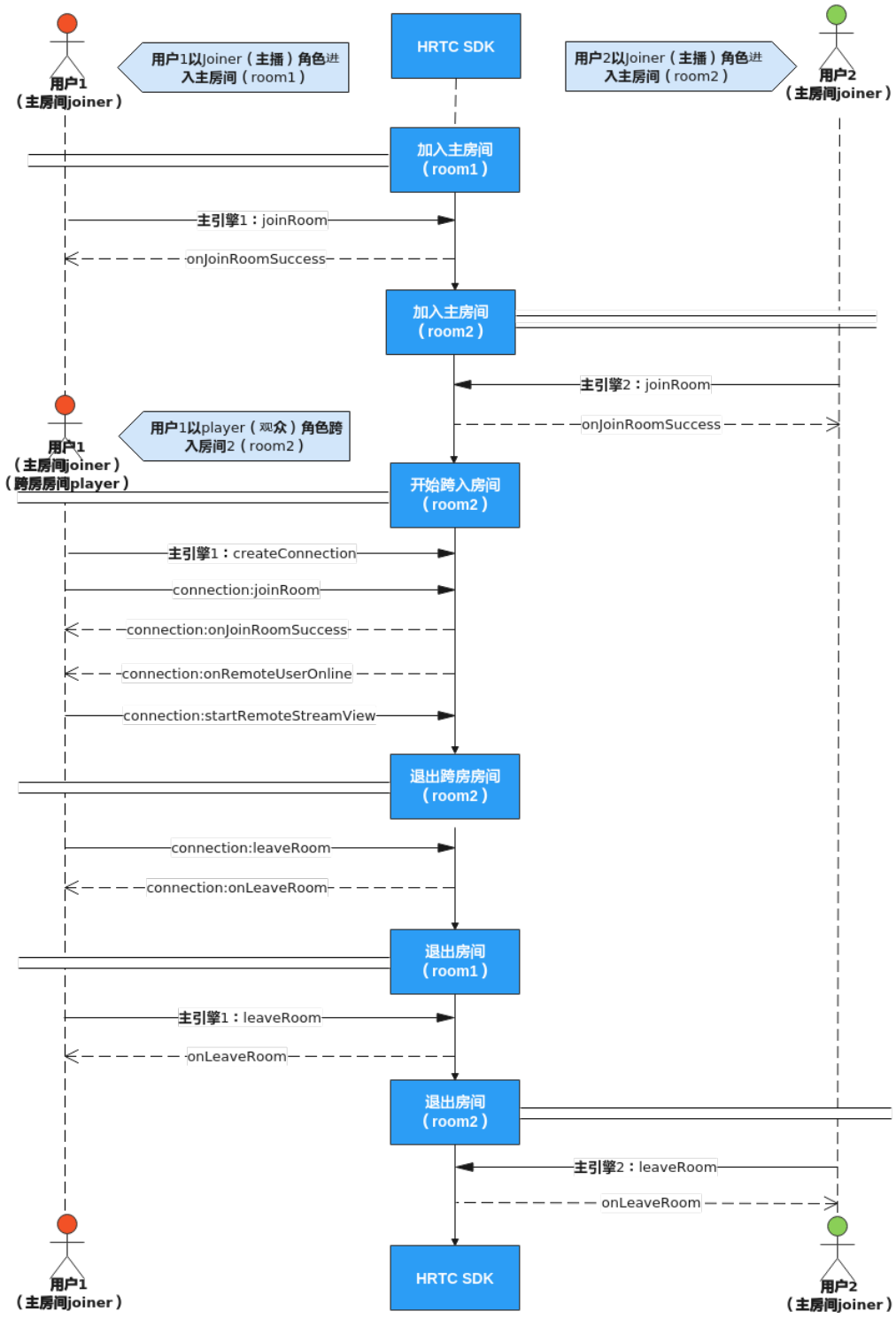
功能描述

跨房间连麦，指主播的媒体流可以同时转发进多个（目前最多支持四个）房间频道，实现主播跨频道与其他主播实时互动的场景。房间中的所有主播可以看见彼此，房间中的观众可以看到所有主播。

注意事项

- 同一时间最多只能创建4个连接对象，每个连接对象对应的房间ID必须互不相同。
- 如果使用connection对象加入房间，则加入房间的房间ID不能和已创建连接对象对应的房间ID相同。
- 同一时间只能以一个JOINER角色加入某一个房间。如果本端在其他房间里的角色是HWRtcRoleJoiner，则需要调用该房间的ChangeUserRole方法将本端在该房间的角色切换为HWRtcRolePlayer后才能以HWRtcRoleJoiner跨入其他房间。如果本端用户是以HWRtcRolePlayer角色加入此跨房房间，则需要调用跨房连接的ChangeUserRole方法将本端的角色切换为HWRtcRoleJoiner后才能发送音频流和视频流。

接口调用流程



实现过程

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入主房间。

2. **创建跨房连接**

当收到远端用户加入房间后会触发IHRTCConnectionEventHandler:onRemoteUserOnline跨房回调，在该回调内用对应远端用户的跨房连接调用startRemoteStreamView方法设置远端窗口并开启收流。

示例代码如下：

```
//创建跨房引擎，room id具有唯一性，不可重复，设置跨房引擎回调
//跨房引擎回调类HwRtcConnectionEventHandler需要继承IHRTCConnectionEventHandler，并重载其回调方法
HwRtcConnectionEventHandler* eventHandler = new HwRtcConnectionEventHandler();
IHRTCConnection* connection = m_pRtcEngine->createConnection(roomId,
(IHRTCConnectionEventHandler*)eventHandler)
```

3. **调用跨房连接的joinRoom加入跨房房间**

   调用跨房连接的joinRoom接口加入房间，其中HRTCJoinParam的authorization鉴权签名字符串需要填入正确的计算值，计算方法请参考**接入鉴权**。

   如果本端需要发送音频或者视频流给跨房房间里的远端用户，则用户的role参数需要设为HRTC_ROLE_TYPE_JOINER，否则可以设为HRTC_ROLE_TYPE_PLAYER，只收流不发流。

   注意：不能同时以**JOINER**角色加入多个房间。

   加入跨房房间成功后会收到跨房回调指针的onJoinRoomSuccess通知，加入房间失败会收到跨房回调指针的onJoinRoomFailure通知。

   示例代码如下：

```
//创建HRTCJoinParam对象
HRTCJoinParam joinRoomParam;
memset(&joinRoomParam, 0, sizeof(HRTCJoinParam));
joinRoomParam.autoSubscribeAudio = true;
joinRoomParam.autoSubscribeVideo = true;
joinRoomParam.userId = "userId";
joinRoomParam.userName = "userName";
joinRoomParam.authorization = authorization; //authorization：鉴权信息，具体生成方法请参见接入鉴权。与加入房间的authorization相同
joinRoomParam.ctime = time;//与加入房间的time相同
joinRoomParam.roomId = "roomId";
joinRoomParam.userRole = HRTC_ROLE_TYPE_PLAYER;
joinRoomParam.sfuType = HRTC_SFU_TYPE_PUBLIC_NETWORK;
joinRoomParam.scenario = HRTC_REMOTE_AUDIO_SUBSCRIBED;
//通过跨房引擎进入房间
connection->joinRoom(joinRoomParam);
```

4. **收远端用户的视频流**

   收到远端用户加入房间后会收到跨房回调onRemoteUserOnline，可以调用该远端用户对应的跨房连接的startRemoteStreamView设置远端窗口并开启收流。

   示例代码如下：

```
//远端用户加入房间后会收到跨房回调
void HwRtcConnectionEventHandler::onRemoteUserOnline(IHRTCConnection* conn, const char* userId, const char* userName) {
    //设置远端窗口并开启收流, viewHandle为显示远端用户视频的窗口句柄
    connection->startRemoteStreamView(userId, viewHandle, HRTC_STREAM_TYPE_HD, true);
}
```

5. **退出跨房房间**

   跨房结束后，调用跨房连接的leaveRoom接口跨房房间。

   示例代码如下：

```
//退出跨房房间
connection->leaveRoom();
```

API 参考

createConnection

connection:joinRoom

connection:onJoinRoomSuccess

connection:onJoinRoomFailure

connection:onRemoteUserOnline

startRemoteStreamView

connection:leaveRoom

2.5 实现音视频通话（Web）

2.5.1 环境准备

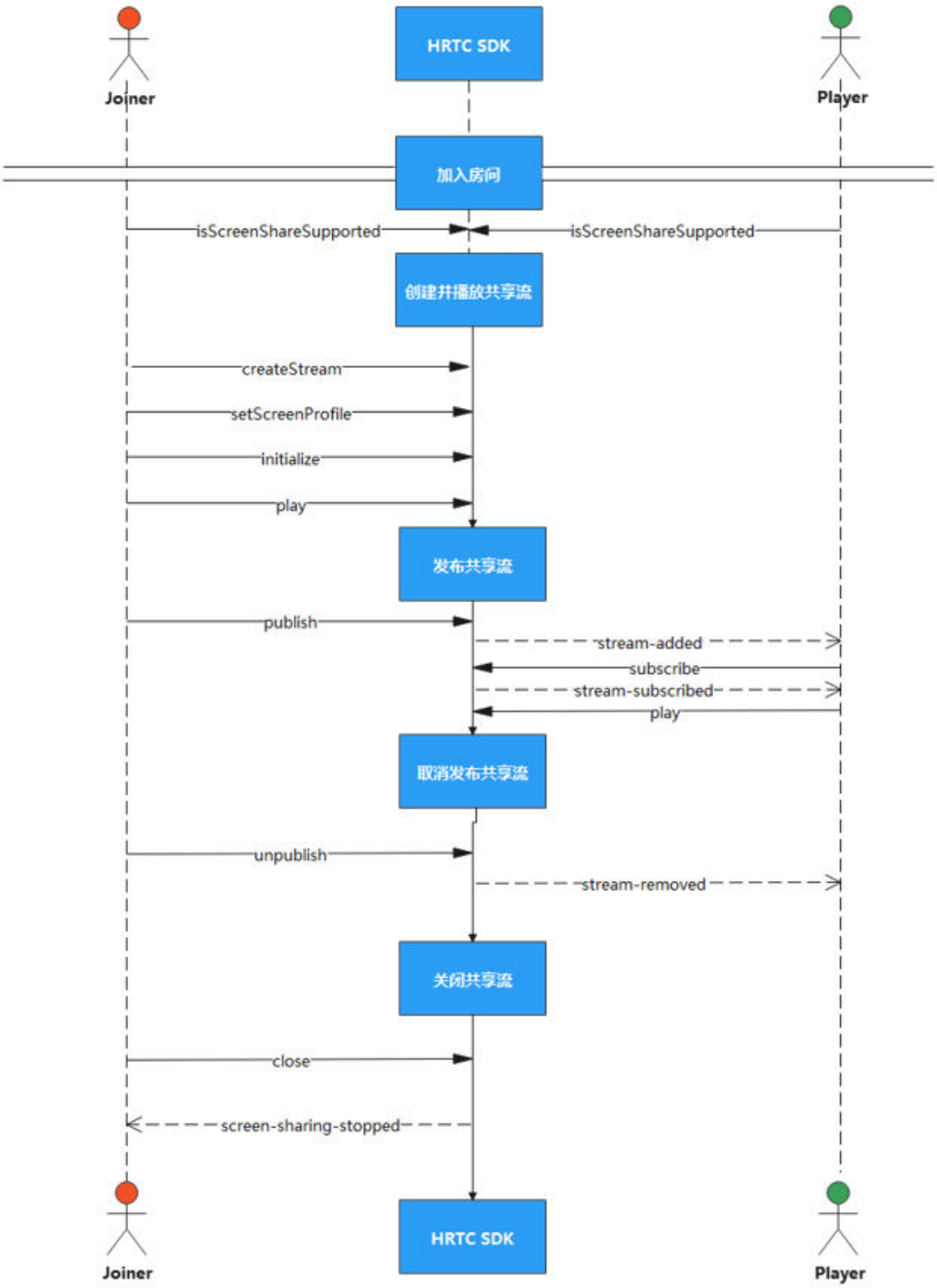
详情请参考开发前准备。

2.5.2 屏幕分享

功能描述

屏幕共享用于在音视频会议中，把一个与会者的屏幕内容，以视频的方式分享给其他与会者。屏幕共享可以共享整个桌面，也可以共享某一个程序窗口。

接口调用流程



实现屏幕共享

1. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤加入房间。

2. **创建并播放共享流**

   入会后调用isScreenShareSupported检测浏览器是否支持共享流。确认支持后，通过调用createStream创建共享流，通过调用setScreenProfile设置共享流的分辨

率，通过调用initialize初始化共享流，通过调用play播放共享流，通过调用bindScreenAudio2RelatedStream设置是否绑定屏幕共享背景音乐至关联流对象。

示例代码如下：

```
// screenAudio 设置是否共享主流音频
let co = {screen: true , screenAudio: false}
// 创建共享流
let localAuxStream = HRTC.createStream(co)
// 设置共享流分辨率
localAuxStream.setScreenProfile('1080p')
// 初始化共享流
localAuxStream.initialize()
.then(() => {   // 播放共享流 c${this.clientIndex}-aux 播放共享流的DOM元素
localAuxStream.play(`c${this.clientIndex}-aux`)
localAuxStream.bindScreenAudio2RelatedStream(this.localStream, screenAudio)})
.catch((error) => {
 console.error(error)
})
```

3. **发布共享流**

   本地播放共享流后，通过调用publish发布本地共享流。

   示例代码如下：

   ```
   this.client.publish(localAuxStream)
   .then(() => {console.info(`发布共享流成功`)})
   .catch((error) => {
    console.error(`发布共享流失败`, error)
   })
   ```

4. **接收远端用户的共享流**

   收到远端用户开启共享流通知stream-added后，通过调用subscribe订阅远端用户的共享流，当共享流订阅成功会收到stream-subscribed回调通知，然后通过调用play在指定的窗口里播放接收到的远端用户的共享流。

   示例代码如下：

   ```
   this.client.on('stream-added', (event: any) => {
   // 流类型为共享流的话
   if (event.stream.getType() === 'auxiliary') {
   // remoteAuxStream 远端共享流
   let remoteAuxStream = event.stream
   this.client.subscribe(remoteAuxStream )
   .then(() => {console.info(`订阅共享流成功`)})
   .catch( (error) =>(console.info(`订阅共享流失败`)))
    }})
   this.client.on('stream-subscribed', (event: any) => {
   // 流类型为共享流的话
   if (event.stream.getType() === 'auxiliary') {
   // c${ clientIndex}-remoteAux 播放共享流的DOM元素
    event.stream.play(`c${ clientIndex}-remoteAux`)
    }
   })
   ```

5. **取消发布共享流**

   共享流发布后，可以通过调用unpublish取消共享流。

   示例代码如下：

   ```
   this.client.unpublish(localAuxStream)
   .then(() => {console.info(`取消发布共享流成功`)})
   .catch((error) => {
   console.error(`取消发布共享流失败`, error)
   })
   ```

6. **停止播放共享流**

   共享流发布后，可以通过调用close停止播放共享流，共享流停止成功后本地会收到screen-sharing-stopped回调信息。

示例代码如下：

```
localAuxStream.close()
localAuxStream.on('screen-sharing-stopped', () => {
 console.log(`屏幕共享停止`)
 localAuxStream = null
})
```

7. **停止接收远端用户的共享流**

   远端用户的共享流取消发布后，本地自动取消订阅，同时会收到stream-removed回调消息。

   示例代码如下：

```
this.client.on('stream-removed', (event: any) => {
 if (event.stream.getType() === 'auxiliary') {
 remoteAuxStream = null
 }
})
```

API 参考

createStream

isScreenShareSupported

setScreenProfile

initialize

play

bindScreenAudio2RelatedStream

publish

subscribe

unpublish

stream-added

stream-subscribed

stream-removed

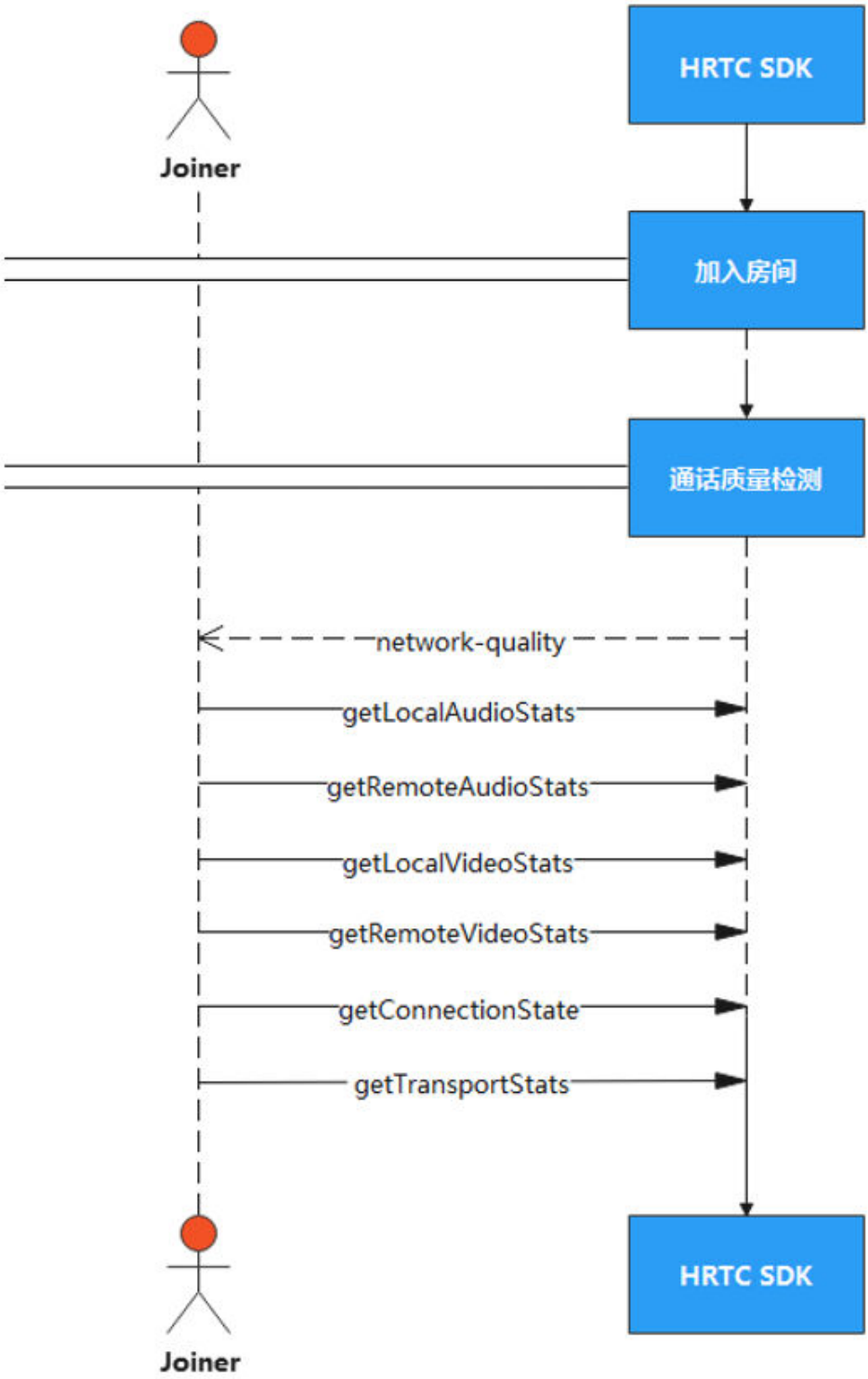
screen-sharing-stopped

2.5.3 通话质量监测

功能描述

加入房间后，SDK上报通话质量相关的回调，上报当前通话的网络质量、本地和远端的音视频统计信息。

接口调用流程



通话质量上报

network-quality网络上下行质量报告事件，用户加入房间后，在网络质量变化的时候会触发一次该事件，报告用户的本地网络上下行质量情况。

示例代码如下：

```
this.client.on('network-quality', (networkQualityInfo) => {
console.info(`network-quality:
uplinkNetworkQuality=${networkQualityInfo.uplinkNetworkQuality},
downlinkNetworkQuality = ${networkQualityInfo.downlinkNetworkQuality}`)
})
```

获取本地音频流统计信息

getLocalAudioStats获取本地设备发送音频流的统计信息。您可以查看当前通话中发送音频的字节数和包数等。

示例代码如下：

```
this.client.getLocalAudioStats().then((stats) => {
 console.info(`getLocalAudioStats: ${stats}`)
})
```

获取远端音频流统计信息

getRemoteAudioStats获取当前通话中远端用户音频流的统计信息。您可以查看远端用户发送的音频流字节数、收包数、丢包率等信息。

示例代码如下：

```
this.client.getRemoteAudioStats().then((stats) => {
 console.info(`getRemoteAudioStats: ${stats}`)
})
```

获取本地视频流统计信息

getLocalVideoStats获取当前本地视频流统计信息，您可以查看本地已发送的字节数、包数、编码帧数，帧数、视频宽度、视频高度等信息。

示例代码如下：

```
this.client.getLocalVideoStats().then((stats) => {
 console.info(`getLocalVideoStats: ${stats}`)
})
```

获取远端视频流统计信息

getRemoteVideoStats获取当前远端视频流统计信息，您可以查看远端用户发送的视频流字节数、包数、编码帧数，帧数、视频宽度、视频高度等信息。

示例代码如下：

```
this.client.getRemoteVideoStats().then((stats) => {
 console.info(`getRemoteVideoStats: ${stats}`)
})
```

获取客户端连接状态

getConnectionState获取客户端的连接状态，分别为：

- CONNECTING：连接建立中。
- CONNECTED：连接已建立。
- RECONNECTING：重新连接中。
- DISCONNECTED：连接已断开。

示例代码如下：

```
console.info(`getConnectingState: ${this.client.getConnectionState()}`)
```

获取当前网络传输状况统计数据

getTransportStats获取当前网络传输状况统计数据，包括已发送字节数、已接收字节数、当前出流码率、当前入流码率等信息，该方法需要publish后调用。

示例代码如下：

```
this.client.getTransportStats().then(
 (rtt) => {
 console.info('###getTransportStats: bytesSent ' + rtt.bytesSent)
 console.info('###getTransportStats: bytesReceived ' + rtt.bytesReceived)
 console.info('###getTransportStats: sendBitrate ' + rtt.sendBitrate)
 console.info('###getTransportStats: recvBitrate ' + rtt.recvBitrate)
 console.info(`getTransportStats: ${rtt.rtt}`)
 },
 (error) => {
 console.info(`getTransportStats: ${error}`)
})
```

API 参考

network-quality

getLocalAudioStats

getRemoteAudioStats

getLocalVideoStats

getRemoteVideoStats

getConnectionState

getTransportStats

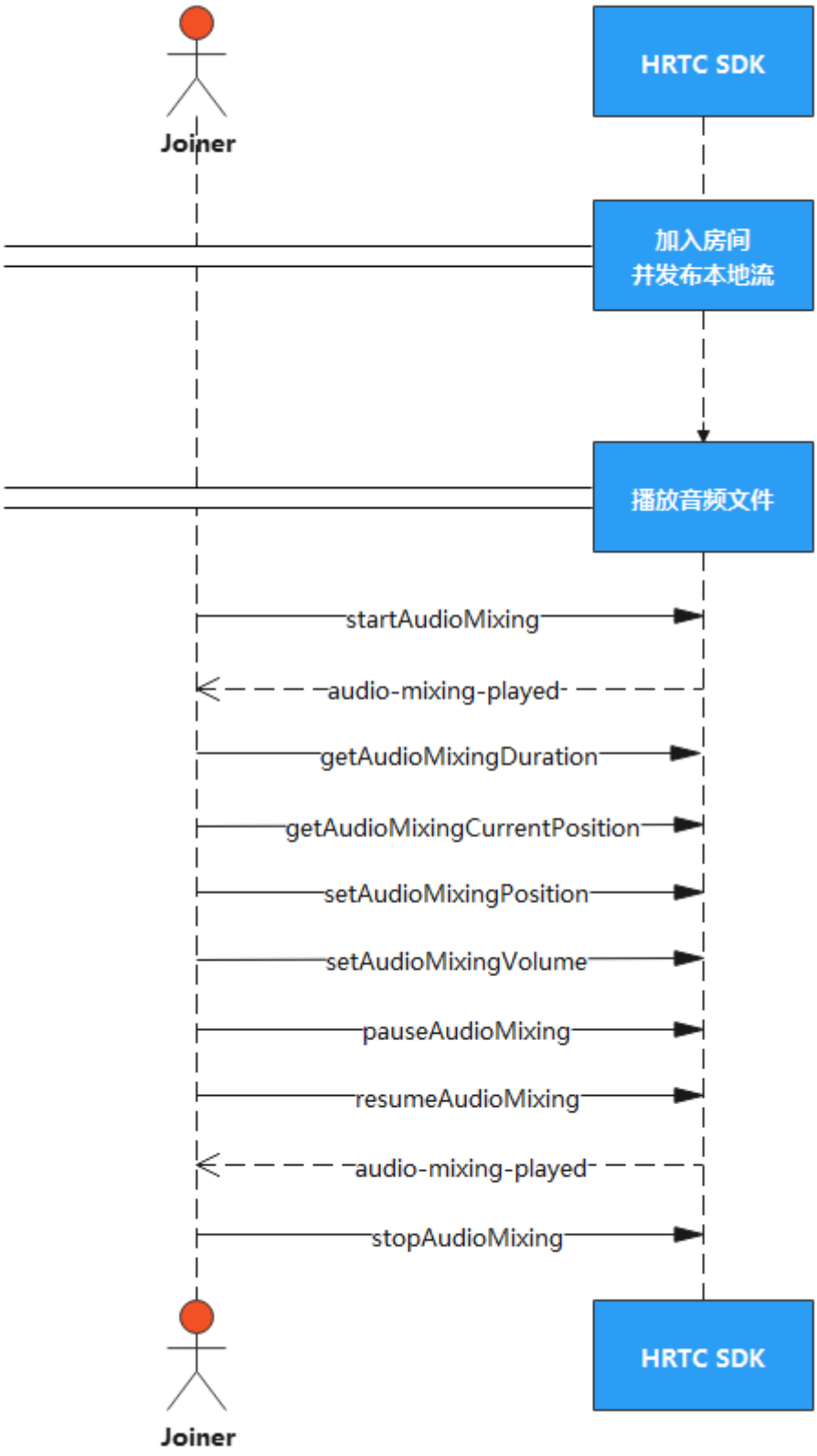
2.5.4 播放音频文件（混音）

功能描述

混音是将音频文件和麦克风音频混合，一般用于播放时长较长的背景音乐或者伴奏。同一时间只能播放一个音频文件，播放给房间内的其他用户听。

支持播放本地或在线音乐文件，文件格式支持播放wav、pcm和单声道mp3音频格式。

接口调用流程



实现过程

1. **加入房间并发布本端主流**

   参考**接口调用流程**中加入房间的时序图步骤加入房间，并发布本地主流。

2. **播放音频文件**

   调用startAudioMixing可以播放一个音频文件，参数设置参见如下示例。同一时刻只能播放一个音频文件。

   ```
   // localStream 本地主流
   localStream.startAudioMixing({
   // filePath 表示在线音频文件的下载路径
   "filePath":"https://***.***.***.***:50007/music.mp3",
   // startTime 表示音频文件开始播放的时间点，默认值为0。
   "startTime":0,
   // replace 表示是否要用音频文件替换本地音频流
   "replace":false,
   // loop 表示是否需要无限循环播放
   "loop":false,
   // repeatCount 表示音频文件循环播放次数
   "repeatCount":0
   })
   ```

3. **设置音频文件音量值**

   音频文件播放过程中，可以调用setAudioMixingVolume设置音频文件播放音量。

   ```
   // volume：音量值
    let volume = 50
    localStream.setAudioMixingVolume (volume)
   ```

4. **获取音频文件总时长**

   音频文件播放成功后，可以调用getAudioMixingDuration获取音频文件总时长，用于刷新界面进度条的总时长。

   ```
   localStream.getAudioMixingDuration()
   ```

5. **获取音频文件播放进度**

   音频文件播放成功后，可以调用getAudioMixingCurrentPosition获取音频文件当前播放时间点。

   ```
   localStream.getAudioMixingCurrentPosition()
   ```

6. **设置音频文件播放位置**

   播放过程中可以调用setAudioMixingPosition设置音频文件播放位置，用于通过拖动进度条改变音频文件的播放位置。

   ```
   localStream.getAudioMixingDuration()
   ```

7. **暂停播放音频文件**

   音频文件播放成功后，可以调用pauseAudioMixing接口暂停播放的音频文件。

   ```
   localStream.pauseAudioMixing()
   ```

8. **恢复播放暂停的音频文件**

   音频文件暂停播放后，可以调用resumeAudioMixing接口恢复播放。

   ```
   localStream.resumeAudioMixing()
   ```

9. **停止播放音频文件**

   音频文件播放成功后，可以调用stopAudioMixing接口停止播放音频文件。

   ```
   localStream.stopAudioMixing()
   ```

10. **音频文件播放开始回调事件**

    音频文件播放开始时，会触发audio-mixing-played回调信息通知App应用。

    ```
    localStream.on('audio-mixing-played', () => {
     console.info('audioMixing: audio-mixing-played')
    })
    ```

11. **音频文件播放结束回调事件**

    音频文件播放结束时，会触发audio-mixing-finished回调信息通知App应用。

```
localStream.on('audio-mixing-finished', () => {
 console.info('audioMixing: audio-mixing-finished')
})
```

API 参考

startAudioMixing

getAudioMixingDuration

getAudioMixingCurrentPosition

setAudioMixingPosition

setAudioMixingVolume

pauseAudioMixing

resumeAudioMixing

stopAudioMixing

2.5.5 切换音频模式

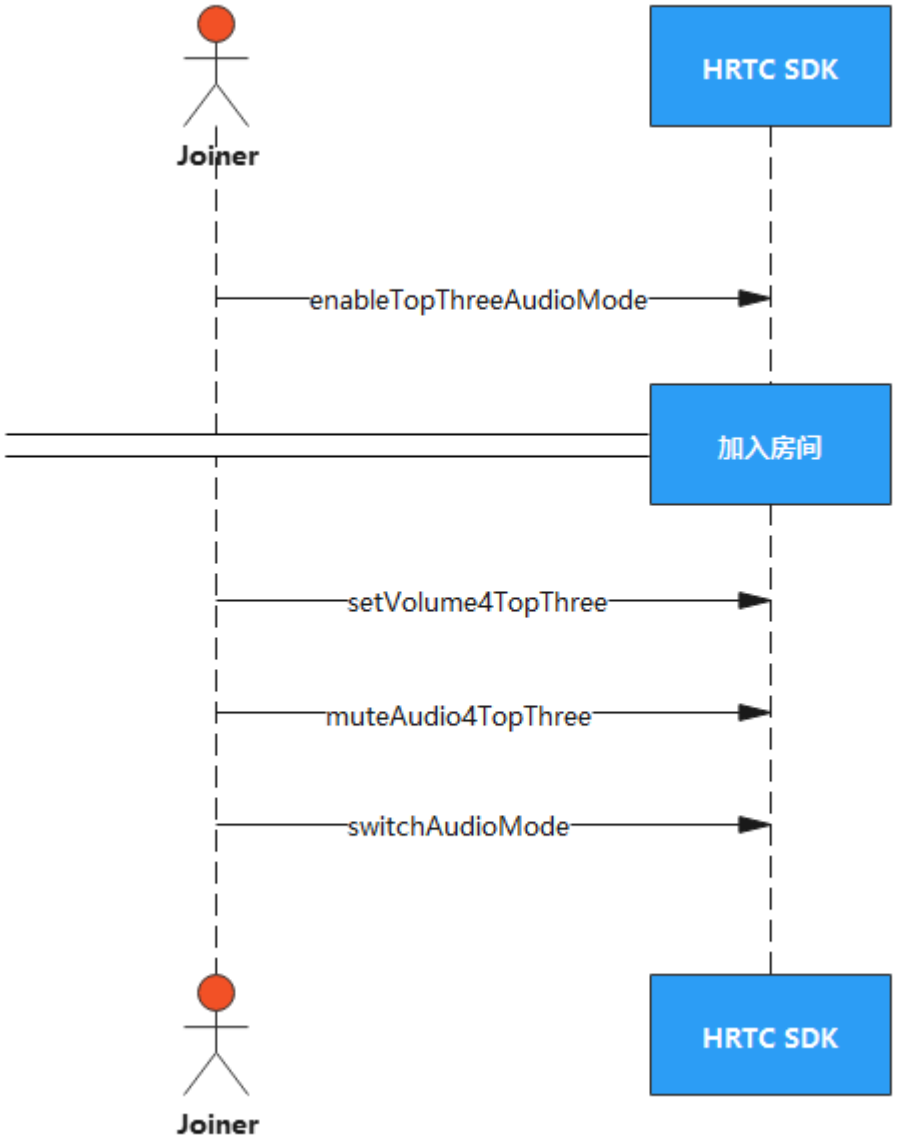
功能描述

用户在入会前可通过调用enableTopThreeAudioMode切换为音频最大三方模式。

会中通过调用switchAudioMode(2)将音频切换为订阅模式。订阅模式下，本地用户必须通过主动订阅远端用户音频流，才可接收该用户音频。

会中通过调用switchAudioMode(3)将音频切换为最大三方模式。最大三方模式下，本地用户不需要单独主动订阅某个远端用户音频流，即可接收当前房间内声音最大的三个用户的音频。

接口调用流程



实现过程

1. **加入房间前设置音频最大三方模式**

   入会前调用enableTopThreeAudioMode设置是否开启音频最大三方模式。true表示开启音频最大三方，false表示不开启。

   示例代码如下：

   ```
   this.client.enableTopThreeAudioMode(true)
   ```

2. **加入房间**

   参考**接口调用流程**中加入房间的时序图步骤。

3. **设置音频最大三方音量值**

调用setVolume4TopThree接口可以设置音频最大三方模式的音量值，取值范围为[0,100]。

示例代码如下：

```
// volume：音量值
let volume = 50
this.client.setVolume4TopThree(volume)
```

4. **开启/禁用音频最大三方模式的音轨**

   调用muteAudio4TopThree接口可以开启/禁用音频最大三方模式的音轨，true表示禁用音频最大三方模式的音轨，false表示开启音频最大三方模式的音轨。

   示例代码如下：

   ```
   this.client.muteAudio4TopThree(true)
   ```

5. **切换音频订阅模式/音频最大三方模式**

   调用switchAudioMode可以切换音频模式。

   当调用switchAudioMode(2)时，表示当前为音频订阅模式；当调用switchAudioMode(3)时，表示当前为音频最大三方模式。

   示例代码如下：

   ```
   this.client.switchAudioMode(2)
   this.client.switchAudioMode(3)
   ```

API 参考

enableTopThreeAudioMode

setVolume4TopThree

muteAudio4TopThree

3 修订记录

表 3-1 表 1 修订记录

发布日期	修订说明
2023-11-30	实时音视频所有客户端类型均增加“环境准备”章节。
2022-09-30	更新“单流录制”和“合流录制”章节的操作内容。
2021-10-30	第一次正式发布。