Zealcomm iRTC JavaScript SDK v4.4.x 开发指南

Zealcomm

修订记录

版本编号	版本日期	修订人	修订内容
0.1	2020.2.15	changjian.sun@zealcomm.com	初稿
0.1.1	2020.2.21	minyi.yang@zealcomm.com	增加监听房间事件
0.1.2	2022.09.21	baorisheng@zealcomm.com	增加媒体录制功能和音频文件发布功能
0.1.3	2.22.09.22	jincheng.shi@zealcomm.cn	增加共享白板功能

目录

1 简介

为了方便开发者使用 Zealcomm iRTC JavaScript SDK，本文档介绍了如何使用 SDK 完成基本媒体流的接收发送的功能，具体包括初始化程序，创建实例，进入房间，监听房间事件，订阅媒体流和发布媒体流、媒体流录制到本地、音频文件发布，使用共享白板等步骤。

2 文档相关内容

本文档主要涉及了如何使用 SDK 完成以下各个功能的简单代码：

- 程序初始化
- 创建 ConferenceClient 实例
- 进入会议房间
- 监听房间事件
- 订阅远端媒体流
- 发布本地媒体流
- 媒体流录制到本地
- 本地音频文件发布
- 共享白板

3 JavaScript SDK 编程指南

3.1 初始化程序

引入必须的 js 文件

```
<script src="js/socket.io.js" type="text/javascript"></script>
<script src="js/irtc.js" type="text/javascript"></script>
```

3.2 创建 ConferenceClient 实例

为了实现 iRTC SDK 的音视频功能，我们需要创建一个 ConferenceClient 实例。

```
let {ConferenceClient} = IRtc.Conference;
var irtcClient = new ConferenceClient();
```

3.3 进入会议房间

完成 ConferenceClient 创建，应用程序就可以用它进入 ZMS 创建的会议房间，进行媒体流交互。

```
irtcClient.join(token)
   .then(function (roominfo) { })
   .catch(function (error) { })
```

加入会议的参数 token 是通过 ZMS 的 HTTP API: **/v1/rooms/:room/tokens** 获得。

加入成功后返回 roominfo（ConferenceInfo），可以通过不同属性获取当前会议的信息。

```
participants：参与者列表
remoteStreams：远端媒体流列表
id：会议 ID
self：当前客户端信息
```

3.4 监听房间事件

完成 ConferenceClient 创建，应用程序可以通过增加监听器来接收房间内各类事件回调

```
irtcClient.addEventListener("streamadded", streamaddedListener);
irtcClient.addEventListener('participantjoined', participantjoinedListener);
irtcClient.addEventListener('messagereceived', messagereceivedListener);
irtcClient.addEventListener("serverdisconnected", serverdisconnectedListener);
```

主要包括下面 4 个房间事件：

- **streamadded:** 在有用户发布新流之后，SDK 会回调这个方法。其中的 stream 代表新发布流，如需要订阅请参考 3.5 订阅远端媒体流。

```
let streamaddedListener = (eve) => {
  let remoteStream = eve.stream;
}
```

- **participantjoined:** 在有新用户加入房间之后，SDK 会回调这个方法。其中的 participant 代表新加入的用户。

```
let participantjoinedListener = (eve) => {
    let participant = eve.participant;
}
```

- **messagereceived:** 在有用户在房间内发送消息时，SDK 会回调这个方法。其中的 msg 代表发送的消息文本，from 代表发生着，to 代表接收者。

```
let  messagereceivedListener = (eve) => {
    let msg = eve.msg;
    let from = eve.from;
    let to = eve.to;
}
```

- **serverdisconnected:** 当 irtcClient 和房间断开连接时，SDK 会回调这个方法。

```
let serverdisconnectedListener = () => {
    console.log('server disconnected');
}
```

3.5 订阅远端媒体流

获取了房间内的远端媒体流列表，应用可以选择订阅并显示远端媒体流：

```
irtcClient.subscribe(stream, option)
   .then((subscription) => { },
         (err) => { })
```

- 订阅时的参数 stream（**RemoteStream**）就是需要订阅的远端媒体流，可以通过 roominfo.remoteStreams 来获得

- 订阅时的参数 options（**SubscribeOptions**）是订阅远端媒体流的设置参数

```
let option = {
   audio: audioOptions,
   video: videoOptions
};
```

audioOption（**AudioSubscriptionConstraints**）和 videoOption（**VideoSubscriptionConstraints**）是相关的订阅音频和订阅视频媒体参数。

- 订阅返回成功后的 subscription（**Subscription**）是对应媒体流的订阅对象；

- 订阅成功后就可以显示 stream，需要在 html 中引入一个 video 标签对象并将其 srcObject 设置为对应流的 mediaStream

```
<video class="screenvideo" autoplay controls muted id="screenvideo">
$('#screenvideo').get(0).srcObject = stream.mediaStream;
```

3.6 发布本地媒体流

除了订阅远端流，conferenceClient 还可以发布本地创建的媒体流：

```
irtcClient.publish(localStream,option)
   .then(function (publication) { })
   .catch(function (error) { })
```

- 发布时的参数 localStream 是本地设备用摄像头创建的媒体流。

首先通过 **MediaStreamFactory.createMediaStream** 和 **StreamConstraints** 创建 **MediaStream**，之后再通过 LocalStream 和 streamSourceInfo 创建需要的本地媒体流：

```
let videoSource = "camera"; //视频源信息。可选参数有"camera", "screen-cast", "file", "mixed" 或 undefined。
let audioSource = "mic"; //音频源信息。可选参数有"mic", "screen-cast", "file", "mixed" 或 undefined.
let audio =  {
   deviceId:undefined,//如果 source 不是"mic"就不用提供。
   source: audioSource,
};
let video = {
   resolution: { width:640, height:480 },
   frameRate: 15,
   deviceId: undefined,// 如果 source 不是"camera"就不用提供。
   source: videoSource,
};
let streamSourceInfo = new StreamSourceInfo(audioSource, videoSource);
let mediaStreamDeviceConstraints = { audio: audio, video: video }
MediaStreamFactory.createMediaStream(mediaStreamDeviceConstraints)
   .then((mediaStream) => {
      localStream = new LocalStream(mediaStream, streamSourceInfo, null)
   }, error => { });
```

- 发布时的参数 option（**PublishOptions**）是发布本地媒体流的设置参数

```
let videoOptions = [{ "codec": { "name":"h264" }, "maxBitrate":800}];
let audioOptions = [{ "codec": { "name":"opus" } }];
let option = { audio: audioOptions, video: videoOptions };
```

audioOption（**AudioEncodingParameters**）和 videoOption（**VideoEncodingParameters**）是相关的发布音频和发布视频媒体参数。

- 发布返回成功后的 Publication 是对应媒体流的发布对象。
- 如要显示本地媒体流，需要在 html 中引入一个 video 标签对象并将其 srcObject 设置为本地流的 mediaStream
-

```
<video class="screenvideo" autoplay controls muted id="localvideo">
$('#localvideo').get(0).srcObject = localStream.mediaStream;
```

3.7 媒体流本地录制功能

通过 Recorder 类 SDK 可以直接把音视频流录制在本地设备。

录制时需要指定音视频流

开始录制

```
//第一个参数：输入的视频流；第二个参数：输入的音频流列，第三个参数：录制参数；
record = new Recorder(stream, audioFroms, options);
record.start();
```

停止录制

```
record.stop();
```

获取本地录制文件(通过浏览器下载，默认输出录制文件名 Data.now() + ".webm")

```
record.download(fileName);
```

3.8 本地音频文件发布功能

SDK 通过 LocalStream 创建本地音频流

从音频文件创建 LocalStream，创建完成后音频就开始播放

```
let audioConstraints = { //用于创建媒体流的音频设置参数
    source: 'raw-file', //定义音频流的音频源类型为文件方式
    file: 'audio-source-file.m4a', //音频文件路径
    loop: true//是否循环
};
let mediaStreamDeviceConstraints = { audio: audioConstraints, video: videoConstraints };
let streamSourceInfo = new StreamSourceInfo('raw-file', videoSource);
MediaStreamFactory.createMediaStream(mediaStreamDeviceConstraints)
.then(mediaStream => { // 返回的 MediaStream 用于创建 LocalStream
          return new LocalStream(mediaStream, streamSourceInfo, { key1: 'value1' }, upgrade);})
.then(localStream => { //返回的 LocalStream 之后可以用于发布
          localStream_ =  localStream; })
.catch(err => {   //处理创建流时的错误异常  });
```

发布音频流

```
client.publish(localStream_, options)
.then(publication => { //返回 publication 给应用程序处理 })
.catch(err => {   //处理发布流时的错误异常  });
```

销毁 LocalStream 后创建的本地文件音频流会一起停止

```
localStream_ = null;
```

3.9 共享白板

SDK 通过 ConferenceClient 可以创建，开启，关闭和删除共享白板

定义白板相关参数

```
let name = ' pdfViewer ';
let uri  = ' https://localhost/uploads/template1'; //白板模板地址
let attributes = { publisher: 'js' };
```

创建白板

第一个参数名称，第二个参数模板地址，第三个参数自定义参数

```
client.createWhiteboard(name, uri, attributes)
 .then(whiteboard => {  //处理白板创建成功事件，返回值是当前创建好的白板 whiteboard  })
 .catch(e => { //处理白板创建失败事件  });
```

开启白板

输入参数创建成功的白板实例

```
client.startWhiteboard(whiteboard)
.then(iframe => {
        //处理开启白板成功事件，返回值是白板对应的 WebView
        $('#irtc-coshow').append(iframe); })
.catch(e => { //处理开启白板失败事件 });
```

关闭白板

输入参数白板实例

```
client.stopWhiteboard(whiteboards.get($('#whiteboards').val()))
.then(whiteboard => { //处理关闭白板成功事件，返回值是白板实例 })
.catch(e => { //处理关闭白板失败事件 });
```

删除白板

输入参数白板实例

```
client.removeWhiteboard(whiteboards.get($('#whiteboards').val()))
.then(whiteboard => { //处理删除白板成功事件 })
.catch(e => { //处理删除白板失败事件 });
```

3.10 其他

以上就是用 iRTC JavaScript SDK 实现基本功能的编程指导说明。

除了 ConfernceClient，对于 RemoteMixedStream，Publication，Subscription，本地录制，本地音频文件发布，共享白板等，也可以增加对应的 EventListener 来监听相关事件。而 LocalStream 的音频，视频源除了摄像头外还可以采用其他媒体源。

对于 SDK 的这些使用方法，请参考 SDK API 文档和 sample app。