音视频管理

音视频管理主要分为 `采集` 与 `播放` 两部分。采集是指使用自己设备的麦克风和摄像头获取自己的音、视频数据，推送到服务端供教室内的其他用户播放，老师可以远程开关对象用户的麦克风、摄像头，即关闭对象的音视频采集；播放则是指播放其他正在发言的用户的音、视频，用户可以选择是否播放发言用户的视频，而音频是默认播放的，不可控制。

为了更方便的对教室内的音视频进行管理，需要对音视频的状态进行监听。音视频状态监听主要包含对当前采集 / 播放状态的监听，其中**音视频用户列表表示当前教室所有正在发言的其他用户（不包含用户自身），监听它的变化是准确播放对象用户视频的前提**。

音视频采集
音视频采集权限检测回调
开启 / 关闭音视频采集、采集视图展示
禁止、允许开启麦克风，一键开关麦克风
音视频采集状态
移动端设备作为外接摄像头
采集设置：视频方向，清晰度，美颜，前后摄像头
水平、垂直镜像

视频播放

播放控制，播放、关闭指定用户的视频，获取视频，显示模式
监听用户音视频开关状态，音视频用户列表变化
音视频用户的播放状态，清晰度，水印等
播放媒体文件
专业小班课上下台、视频窗口位置同步

举手发言
学生举手、取消举手，老师处理举手申请
邀请/强制发言的发起和处理
禁止、允许举手

直播间音视频设置
是否支持后台音频播放
是否支持后台采集
是否播放视频静音
WebRTC 教室回调

音视频采集

音视频采集功能由 `BJLRecordingVM` 管理。

1. 音视频采集控制

对于老师，开启采集有两个前提条件：进入教室成功和处于上课状态。

进入教室成功通过监听到 `BJLRoom` 的 `enterRoomSuccess` 方法得知，上课状态则通过监听 `BJLRoomVM` 的 `liveStarted` 方法获取。

对于学生，在大班课场景下，还需要处于发言状态才可以开启音视频采集，参考举手发言部分的内容。

- 音视频采集权限检测。

SDK 提供了基础的麦克风和摄像头权限检测的弹窗，在使用前会检查权限状态，如果未授权，会回调弹窗给 UI 层处理。此外，如果需要，UI 层也可以直接自定义权限检测的相关弹窗等功能，在使用到相关功能前提前检测。

```
/**
 检测麦克风和摄像头权限回调
 #disussion 在需要开启音视频时，检测对应权限之后的回调，
 #disussion microphone -> YES 需要麦克风权限，camera -> YES 需要摄像头权限，
 #disussion granted -> YES 已授权，可以不做处理，granted -> NO 未授权，抛出 alert，需要展示
 */
@property (nonatomic, nullable) void (^checkMicrophoneAndCameraAccessCallback)(BOOL microphone, BOOL camera, BOOL granted, UIAlertController * _Nullable alert);
/** 检测权限的弹窗操作完成后的回调，用来队列化处理麦克风和摄像头都未授权时弹出的二个弹窗，但是未决定授权状态的弹窗不会通过此处回调 */
@property (nonatomic, nullable) void (^checkMicrophoneAndCameraAccessActionComple
(void);

// example：设置权限检测回调，在监听到enterRoomSuccess之后
```

```
__block UIAlertController *alertController = nil;
[self.room.recordingVM setCheckMicrophoneAndCameraAccessCallback:^(B
 microphone, BOOL camera, BOOL granted, UIAlertController * _Nullable alert) {
    bjl_strongify(self);
    if (granted || !alert) {
        return;
    }
    // 弹出请求授权的提示
    if (self.presentedViewController) {
        if (self.presentedViewController == alertController && alert != alertController) {
            [self.room.recordingVM setCheckMicrophoneAndCameraAccessActionComp
                bjl_strongify(self);

self.room.recordingVM.checkMicrophoneAndCamera
 = nil;
                alertController = alert;
                if (self.presentedViewController) {
                    [self.presentedViewController bjl_dismissAnimated:YES completion:nil];
                }
                [self presentViewController:alert animated:YES completion:nil];
            }];
        }
        else {
            alertController = alert;
            [self.presentedViewController bjl_dismissAnimated:YES completion:nil];
            [self presentViewController:alert animated:YES completion:nil];
        }
    }
```

```
    else {
        alertController = alert;
        [self presentViewController:alert animated:YES completion:nil];
    }
}];
```

- 添加采集视图。

采集视图的实例进入教室初始化之后，在销毁教室前会一直存在，可以根据具体的需求和场景控制其显示、隐藏、调整位置、尺寸等。在开启视频，监听到 `recordingVideo` 为 YES 前，采集视图是纯黑的，如果需要显示的话，可以遮盖一张占位图。

```
// example：将 BJLRoom 的 recordingView 添加到当前 viewController 的对应视图
[self.recordingView addSubview:self.room.recordingView];
```

2. 音视频采集状态

通过监听 `self.room.recordingVM` 的属性变化及方法调用来实现。

- 关键属性监听。

```
// 音频采集开关状态，默认 NO
@property (nonatomic, readonly) BOOL recordingAudio;
// 视频采集开关状态，默认 NO
@property (nonatomic, readonly) BOOL recordingVideo;
/** 本地视频预览画面是否在加载中。不支持 AVSDK、百家云 底层 */
```

```
@property (nonatomic, readonly) BOOL localVideoLoading;
// 音频输入级别 [0.0 - 1.0]
@property (nonatomic, readonly) CGFloat inputVolumeLevel;
// 视频采集宽高比，和视频分辨率有关，不低于 720p 时为 16:9，否则为 4:3
@property (nonatomic, readonly) CGFloat inputVideoAspectRatio;
// 是否禁止当前用户打开音频，默认 NO
@property (nonatomic, readonly) BOOL forbidRecordingAudio;
// 是否禁止所有人打开音频，默认 NO
@property (nonatomic, readonly) BOOL forbidAllRecordingAudio;
```

以下监听均需要在教室进入成功之后监听有效：

```
// example：监听视频采集（摄像头）开关状态
[self bjl_kvo:BJLMakeProperty(self.room.recordingVM, recordingVideo)
      filter:^BOOL(NSNumber * _Nullable now, NSNumber * _Nullable old, BJLPropertyChange * _Nullable change) {
    return now.boolValue != old.boolValue;
}
     observer:^BOOL(NSNumber * _Nullable now, NSNumber * _Nullable old, BJLPropertyChange * _Nullable change) {
    // bjl_strongify(self);
    NSLog(@"摄像头已%@", now.boolValue ? @"打开" : @"关闭");
    return YES;
}];
```

```
// example：监听麦克风音频输入级别 inputVolumeLevel
[self bjl_kvo:BJLMakeProperty(self.room.recordingVM, inputVolumeLevel)
    options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld
    filter:^BOOL(NSNumber * _Nullable value, NSNumber * _Nullable oldValue, BJLPropertyChange * _Nullable change) {
  // 音量变化超过一定程度才触发
  return ABS(round(oldValue.doubleValue * 10) - round(value.doubleValue * 10)) >= 1.0;
}
    observer:^BOOL(NSNumber * _Nullable value, NSNumber * _Nullable oldValue, BJLPropertyChange * _Nullable change) {
  NSLog(@"current input volume level:%f", value.doubleValue);
  return YES;
}];
```

- 采集音视频开关。
- 使用 `setRecordingAudio:recordingVideo:` 开关音视频。

```
/**
 开关音视频 (需监听到 进入教室成功 和 处于上课状态，身份为学生则还需要处于发言状态）
 #param recordingAudio YES：打开音频采集，NO：关闭音频采集
 #param recordingVideo YES：打开视频采集，NO：关闭视频采集
 #discussion 上层自行检查麦克风、摄像头开关权限
```

```
 #discussion 上层可通过 `BJLSpeakingRequestVM` 实现学生发言需要举手的逻辑
 #return BJLError:
 BJLErrorCode_invalidCalling    错误调用，以下情况下开启音视频、在音频教室开启摄像头均会返回此错误
 登录用户分组 ID 不为 0，参考 `room.loginUser.groupID`
 非上课状态，参考 `room.roomVM.liveStarted`
 教室禁止打开音频，参考 `self.forbidRecordingAudio`
 音频禁止打开视频，参考 `featureConfig.mediaLimit`
 */
BJLError *error = [self.room.recordingVM setRecordingAudio:YES recordingVideo:YES];
if (error) {
  NSString *errorMessage = error.localizedFailureReason ?: error.localizedDescription;
  NSLog(@"%td-%@", error.code, errorMessage);
}
```

- 音视频采集请求被服务端拒绝。

调用 `setRecordingAudio:recordingVideo:` 方法开启音视频采集时，可能因为教室内发言用户人数达到上限而被服务器拒绝，可通过监听 `recordingDidDeny` 方法给出错误提示，发言用户人数上限可联系百家云后台进行配置调整。

```
 开启音视频被服务端拒绝
 #discussion 可能因为上麦路数达到上限 */
- (BJLObservable)recordingDidDeny;

/** example：开启音视频采集失败的提示 */
```

```
[self bjl_observe:BJLMakeMethod(self.room.recordingVM, recordingDidDeny)
        observer:^BOOL {
    // bjl_strongify(self);
    NSLog(@"服务器拒绝发布音视频，音视频并发已达上限");
    return YES;
}];
```

- 老师、助教：远程开关学生音、视频。

```
/**
 老师: 远程开关学生音、视频
 #param user 对象用户，不能是老师
 #param audioOn YES：打开音频采集，NO：关闭音频采集
 #param videoOn YES：打开视频采集，NO：关闭视频采集
 #discussion 打开音频、视频会导致对方发言状态开启
 #discussion 同时关闭音频、视频会导致对方发言状态终止
 @see `speakingRequestVM.speakingEnabled`
 #return BJLError:
 BJLErrorCode_invalidArguments  错误参数；
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限。
 */
 BJLError *error = [self.room.recordingVM remoteChangeRecordingWithUser:self.user audioOn:self.user.audioOn videoOn:on];
```

```
/**
```

```
 老师: 远程开启学生音、视频被自动拒绝，因为上麦路数达到上限
 #param user    开启失败的学生
 */
[self bjl_observe:BJLMakeMethod(self.room.recordingVM, remoteChangeRecordingDidDenyForUser:)
        observer:^BOOL(BJLUser *user) {
    // bjl_strongify(self);
    NSLog(@"服务器拒绝强制 %@ 发言，音视频并发已达上限", user.name);
    return YES;
}];
```

- 老师、助教：开启/关闭 全体禁音。

```
/**
 老师: 设置全体禁音状态
 #param forbidAll YES：全体禁音，NO：取消禁音
 #discussion 设置成功后修改 `forbidAllRecordingAudio`、`forbidRecordingAudio`
 #return BJLError:
 BJLErrorCode_invalidUserRole    错误权限，要求老师或助教权限
 */
[self.room.recordingVM sendForbidAllRecordingAudio:YES];
```

- 老师、助教：开关全体学生麦克风。

```
/**
 开关全体学生麦克风
 #param mute YES：关闭，NO：打开
 #return BJLError
```

```
5. */
6. [self.room.recordingVM
updateAllRecordingAudioMute:YES];
```

- 老师、助教：设置全体/单个用户视频镜像模式。

```
1. // 当前用户推流镜像模式
2. @property (nonatomic, readonly)
BJLEncoderMirrorMode
currentUserVideoEncoderMirrorMode;
3.
4. // example：设置所有开启视频的用户的推流镜像模式
5. BJLError *error = [self.room.recordingVM
updateVideoEncoderMirrorModeForAllPlayingUser:m
6. // example：设置指定用户的推流镜像模式
7. BJLError *error = [self.room.recordingVM
updateVideoEncoderMirrorMode:mode
forUser:self.room.loginUser];
```

- 关键方法监听。

```
1. // example：监听 摄像头/麦克风 被老师远程开关
2. [self
bjl_observe:BJLMakeMethod(self.room.recordingVM,

recordingDidRemoteChangedRecordingAudio:recorc
3.         observer:
(BJLMethodObserver)^BOOL(BOOL
recordingAudio, BOOL recordingVideo, BOOL
recordingAudioChanged, BOOL
recordingVideoChanged) {
4.     // bjl_strongify(self);
5.     NSString *message = @"";
6.     if (recordingAudioChanged) {
```

```
        message = recordingAudio ? @"老师开启了你的麦克风" : @"老师关闭了你的麦克风";
    }
    else if (recordingVideoChanged) {
        message = recordingVideo ? @"老师开启了你的摄像头" : @"老师关闭了你的摄像头";
    }
    return YES;
}];
```

```
// example：监听一键开关麦克风
[self bjl_observe:BJLMakeMethod(self.room.recordingVM, didUpadateAllRecordingAudioMute:)
         observer:(BJLMethodObserver)^BOOL(BOOL mute) {
    // bjl_strongify(self);
    if (mute) {
        NSLog(@"老师已关闭全体学生的麦克风");
    }
    else {
        NSLog(@"老师已开启全体学生的麦克风");
    }
    return YES;
}];
```

```
// example：监听音频本地 PCM 数据
    [self bjl_observe:BJLMakeMethod(self.room.recordingVM, onAudioLocalProcessedAudioFrame:)
             observer:^BOOL(BJLCustomAudioFrame *frame) {
        return YES;
    }];
```

```
    [self bjl_observe:BJLMakeMethod(self.room.recordingVM, onAudioCapturedRawAudioFrame:)
            observer:^BOOL(BJLCustomAudioFrame *frame) {
        return YES;
    }];
// 注意：如果想开启 PCM 数据回调，需要先开启 enablePCMData 属性
self.room.recordingVM.enablePCMData = YES;
```

3. 移动端设备作为外接摄像头采集

2.11.0 版本开始支持外接设备作为摄像头，仅主讲人支持使用，作为摄像头的设备可以通过非参加码的方式进入教室，作为主讲人的摄像头画面采集。

- 获取外接设备进入教室的链接和二维码。

```
    [self.room.recordingVM requestAsCameraDataWithCompletion:^(NSString * _Nullable urlString, UIImage * _Nullable image, BJLError * _Nullable error) {
        bjl_strongify(self);
        if (image) {
      // 显示外接设备进入教室的二维码
        }
    }];
```

- 外接设备进入教室。

可以直接使用获取到的外接设备的进教室链接进入教室，或者提供二维码，使用扫码解析后的链接进入教室。

```
/**
 通过链接创建教室，创建教室可能失败，返回 nil
 #param string        一般为 APP 拉起链接
 #return              教室或者 nil
 */
+ (nullable __kindof instancetype)roomWithURLString:(NSString *)string;
```

- 外接设备控制。

主讲人可以获取当前是否存在外接设备，当存在外接设备时，开关自己的摄像头会自动转换成控制外接设备的摄像头。主讲人也可以随时停止外接设备的使用。其他观看的用户正常获取主讲人的视频画面即可。

```
/** 当前用户是否存在外接移动端摄像头 */
@property (nonatomic, readonly) BOOL hasAsCameraUser;

/** 停止外接摄像头的使用，停止完毕后回调 completion，否则不回调 */
- (nullable BJLError *)stopAsCameraUserCompletion:(void (^__nullable)(void))completion;
```

4. 音视频采集设置

- 禁止采集声音。

```
/**
 学生: 是否禁止当前用户打开音频 - 个人实际状态
 #discussion 用于判断当前用户是否能打开音频
 #discussion 参考 `forbidAllRecordingAudio`
```

```
 */
@property (nonatomic, readonly) BOOL forbidRecordingAudio;
```

```
/**
 是否禁止所有人打开音频 - 全局开关状态
 #discussion 用于判断教室内开关状态
 #discussion 如果学生正在采集音频，收到此事件时会被自动关闭
 #discussion 课程类型为小班课、新版小班课、双师课时可用，参考 `room.roomInfo.roomType`、`BJLRoomType`
 #discussion 1. 当老师禁止所有人打开音频时，`forbidAllRecordingAudio` 和 `forbidRecordingAudio` 同时被设置为 YES，
 #discussion 2. 当老师取消禁止所有人打开音频时，`forbidAllRecordingAudio` 和 `forbidRecordingAudio` 同时被设置为 NO，
 #discussion 3. 当老师邀请/强制当前用户发言时，`forbidAllRecordingAudio` 被设置成 NO，`forbidRecordingAudio` 依然是 YES，
 #discussion 4. 当老师取消邀请/强制结束当前用户发言时，`forbidAllRecordingAudio` 会被设置为与 `forbidRecordingAudio` 一样的取值
 */
@property (nonatomic, readonly) BOOL forbidAllRecordingAudio;
```

```
/** 用户是否在录制屏幕，iOS11以上生效，iOS11以下一直是NO。如果当前正在使用App自己的屏幕分享功能，则此变量也是一直为NO */
@property(nonatomic, readonly) BOOL screenCaptured;
```

```
/**
 老师: 设置全体禁音状态
 #param forbidAll YES：全体禁音，NO：取消禁音
 #discussion 设置成功后修改 `forbidAllRecordingAudio`、`forbidRecordingAudio`
 #return BJLError:
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限
 */
BJLError *error = [self.room.recordingVM sendForbidAllRecordingAudio:sender.on];
```

- 切换摄像头。

```
/** 当前使用的摄像头，默认使用前置摄像头 */
@property (nonatomic, readonly) BOOL usingRearCamera; // NO: Front, YES Rear(iSight)
// example：切换前、后置摄像头
[self.room.recordingVM updateUsingRearCamera:!self.room.recordingVM.us
```

- 视频采集方向设置。

```
/** 视频采集模式，默认采集横屏画面 */
@property (nonatomic) BJLVideoRecordingOrientation videoRecordingOrientation;
// example：设置为竖屏采集模式
self.room.recordingVM.videoRecordingOrientation = BJLVideoRecordingOrientation_alwaysPortrait;
```

- 视频采集画面显示模式设置。

```
/** 采集画面显示模式，默认BJLVideoContentMode_aspectFit */
@property (nonatomic) BJLVideoContentMode videoContentMode;

// example: 设置为 fill 的铺满模式
self.room.recordingVM.videoContentMode = BJLVideoContentMode_aspectFill;
```

- 清晰度设置。

```
/** 视频采集清晰度，默认标清 */
@property (nonatomic, readonly) BJLVideoDefinition videoDefinition;

/**
 改变视频清晰度
 #param videoDefinition 清晰度
 #return BJLError:
 BJLErrorCode_invalidCalling  错误调用
 */
 BJLError *error = [self.room.recordingVM updateVideoDefinition:BJLVideoDefinition_high];
```

- 美颜设置。

```
/// 设置磨皮级别，取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显。 仅BRTC底层有效
- (nullable BJLError *)updateBeautyLevel:(float)level;
@property (nonatomic, readonly) float beautyLevel;

```

```
/// 设置美白级别，取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显。 仅BRTC底层有效
- (nullable BJLError *)updateWhitenessLevel:(float)level;
@property (nonatomic, readonly) float whitenessLevel;
```

- 屏幕共享

```
/** 当前是否在进行屏幕共享 */
@property (nonatomic, readonly) BOOL screenSharing;

/**
 开启屏幕共享
 #param appGroup 屏幕分享主进程所属的 group
 #param screenShareBlock 屏幕分享结果回调
 #return BJLError:
 BJLErrorCode_invalidCalling  错误调用
 */
- (void)startScreenShareWithAppGroup:(NSString *)appGroup resultBlock:(nullable void(^)(BJLError * _Nullable error))screenShareBlock API_AVAILABLE(ios(14.0));

/**
 关闭屏幕共享
 #return BJLError:
 BJLErrorCode_invalidCalling  错误调用
 */
- (nullable BJLError *)stopScreenShare API_AVAILABLE(ios(14.0));

- (BJLObservable)onScreenCaptureStarted;
```

```
- (BJLObservable)onScreenCapturePaused:(NSInteger)reason;
- (BJLObservable)onScreenCaptureResumed:(NSInteger)reason;
- (BJLObservable)onScreenCaptureStoped:(NSInteger)reason;
```

- 推流状态监听

```
/** 推流重试中 */
- (BJLObservable)republishing;

/** 推流失败 */
- (BJLObservable)publishFailed;

example:
 // webRTC 推流重试提示
[self bjl_observe:BJLMakeMethod(self.room.recordingVM, republishing)
            observer:^BOOL {
            bjl_strongify(self);
            [self showProgressHUDWithText:BJLLocalizedString(@"音视频推送失败，自动重试中")];
            return YES;
}];

[self bjl_observe:BJLMakeMethod(self.room.recordingVM, publishFailed)
            observer:^BOOL {
            bjl_strongify(self);
        [self showProgressHUDWithText:BJLLocalizedString(@"音
```

```
视频推送失败，请重试")];
            return YES;
}];
```

音视频播放

音视频播放功能由 `BJLPlayingVM` 管理。

1. 音视频播放要点说明

- 直播教室中，一个用户可能同时在推送多路音视频流，也就是说，`BJLOnlineUserVM` 的 `onlineUsers` 数组中的一个 `BJLUser` 实例，在 `BJLPlayingVM` 中可能对应到多个 `BJLMediaUser` 实例，他们的 `ID` 、 `number` 等身份信息相同，区别在于 `BJLUser` 标识一个用户身份，而 `BJLMediaUser` 标识用户的一路音视频流、包含这路流的各项信息。

- `BJLMediaUser` 标识用户的一路音视频流，它的 `mediaSource` 属性表示该音视频流的类型，参考 `BJLMediaSouce` ； `cameraType` 属性主要用于普通大班课， `BJLCameraType_main` 表示在大班课里需要占据主摄像头位置的音视频流，这种流的 `mediaSource` 为 `BJLMediaSource_mainCamera` 、 `BJLMediaSource_screenShare` 或 `BJLMediaSource_mediaFile` ； `BJLCameraType_extra` 表示在大班课里需要占据扩展摄像头位置的音视频流，这种流的 `mediaSource` 为 `BJLMediaSource_extraCamera` 或 `BJLMediaSource_extraScreenShare` 。

多路流模板中用户音视频流对应关系如下：

视频源类型	含义

BJLMediaSourcemainCamera	主摄像头采集
BJLMediaSource_screenShare	屏幕共享
BJLMediaSource_mediaFile	媒体文件播放
BJLMediaSource_extraCamera	辅助摄像头采集
BJLMediaSource_extraScreenShare	辅助摄像头屏幕共享
BJLMediaSource_extraMediaFile	辅助摄像头媒体文件

对于一个 `BJLUser` 实例，在 `BJLPlayingVM` 中可能对应多个 `BJLMediaUser` 实例，每一个实例对应该用户的一路音视频流，它们之间 `ID` 、 `number` 等用户身份信息相同，通过 `mediaID` 和 `mediaSource` 区分。

- 在 `WebRTC` 教室（通过 `self.room.featureConfig.isWebRTC` 判断）中， `mediaSource` 为 `BJLMediaSource_mainCamera` 的 `BJLMediaUser` 实例存储于 `BJLPlayingVM` 的 `playingUsers` 中，每个发言用户在数组中最多存在一个实例；其它视频源类型的实例存储于 `BJLPlayingVM` 的 `extraPlayingUsers` 中，每个发言用户在数组中可能存在多个实例。

- 在非 `WebRTC` 教室中， `mediaSource` 为 `BJLMediaSource_mainCamera` 、 `BJLMediaSource_screenShare` 、 `BJLMediaSource_mediaFile` 的 `BJLMediaUser` 实例存储于 `BJLPlayingVM` 的 `playingUsers` 中，每个发言用户在数组中最多只存在一个实例； `mediaSource` 为 `BJLMediaSource_extraCamera` 或 `BJLMediaSource_extraScreenShare` 的 `BJLMediaSource` 实例存储于 `BJLPlayingVM` 的 `extraPlayingUsers` 中，每个发言用户在数组可能存在多个实例。

- 特别的，在 `WebRTC` 的大班课教室，用户未上台参与互动的情况下，教室内的视频流默认配置成合流展示，当前音视频播放是否使用合流可以参考 `BJLPlayingVM` 的 `playMixedVideo` 。合流状态下只有一个主讲人的音视频用户，使用的是 TCP 方式从 CDN 节点拉流，视频画面是以主讲人为主视频画面，其他视频用户拼成的视频画面。

2. 监听播放信息

音视频信息通过监听 `room.playingVM` 的属性变化获取。

- 监听音视频用户列表。

`BJLPlayingVM` 的 `playingUsers` 属性表示当前正在推送音频、视频的用户列表（不包含用户自身），它是随时会发生变化的，因此**不要采用直接取值的方法获取列表**，而应该监听列表的变化，即时获取最新列表，便于播放对应用户视频。

监听音视频用户列表的变化可以通过监听 `BJLPlayingVM` 的属性 `playingUsers` 的变化来实现：

```
/**
 音视频用户列表
 #discussion 包含教室内推送主音视频流的用户，数组内 BJLMediaUser 实例的音视频信息为主音视频流的信息，每个用户在 playingUsers 中只有一个 BJLMediaUser 实例
 #discussion 在 webRTC 教室中，数组内的 BJLMediaUser 实例的 mediaSource 为 BJLMediaSource_mainCamera
 #discussion 在非 webRTC 教室中，数组内的 BJLMediaUser 实例的 mediaSource 为 BJLMediaSource_mainCamera、BJLMediaSource_screenShare 或 BJLMediaSource_mediaFile
```

```
6. #discussion 所有用户的音频会自动播放，视频需要调用 `updatePlayingUserWithID:videoOn:mediaSource:` 打开或者通过 `autoPlayVideoBlock` 控制打开
7. #discussion SDK 会处理音视频打断、恢复、前后台切换等情况
8. */
9. @property (nonatomic, readonly, copy, nullable) NSArray<BJLMediaUser *> *playingUsers;
```

```
1. // example：监听 playingUsers 变化，获取实时信息
2. [self bjl_kvo:BJLMakeProperty(self.room.playingVM, playingUsers)
3.     observer:^BOOL(id _Nullable value, id _Nullable oldValue, BJLPropertyChange * _Nullable change) {
4.     // bjl_strongify(self);
5.     NSLog(@"playing users changed");
6.     return YES;
7. }];
```

- 监听扩展音视频流用户列表。

`BJLPlayingVM` 的 `extraPlayingUsers` 属性表示正在推送主摄像头之外的扩展音视频流的用户列表（不包含用户自身），与 `playingUsers` 类似，是随时变化的，也需要通过监听来即时获取信息。

```
1. /**
2. 扩展音视频流用户列表
3. #discussion 包含教室内推送扩展音视频流的用户，音视频信息为扩展音视频流的信息，每个用户在 extraPlayingUsers 中可以有多个 BJLMediaUser 实例
```

```
 #discussion 在 webRTC 教室中，数组内的 BJLMediaUser 实例的 mediaSource 为除 BJLMediaSource_mainCamera 之外的音视频流类型
 #discussion 在非 webRTC 教室中，数组内 BJLMediaUser 实例的 mediaSource 为 BJLMediaSource_extraCamera 或 BJLMediaSource_extraScreenShare
 #discussion 所有用户的音频会自动播放，视频需要调用 `updatePlayingUserWithID:videoOn:mediaSource:` 打开或者通过 `autoPlayVideoBlock` 控制打开
 #discussion 打开了扩展音视频流的用户将同时包含在 playingUsers 和 extraPlayingUsers 中，但两个列表中的 BJLMediaUser 实例的音视频信息不同
 */
@property (nonatomic, readonly, copy, nullable) NSArray<BJLMediaUser *> *extraPlayingUsers;
```

```
// example：监听 extraPlayingUsers 变化，获取实时信息
[self bjl_kvo:BJLMakeProperty(self.room.playingVM, extraPlayingUsers)
    observer:^BOOL(id _Nullable value, id _Nullable oldValue, BJLPropertyChange * _Nullable change) {
   // bjl_strongify(self);
   NSLog(@"extra playing users changed");
   return YES;
}];
```

- 监听当前正在播放的视频对应的用户列表。

`BJLPlayingVM` 的 `videoPlayingUsers` 属性表示当前正在播放的视频用户列表，如果当前正在播放对方用户的视频，即通

过调用 `updatePlayingUserWithID:videoOn:mediaSource:` 方法播放了用户的视频画面，或者通过 `autoPlayVideoBlock` 设置了自动播放音视频用户的视频画面时，则该视频流对应的 `BJLMediaUser` 实例会被包含在 `videoPlayingUsers` 中。

```
/**
 正在播放的视频用户
 #discussion 数组内元素包含在 `playingUsers`、`extraPlayingUsers` 之中，在当前打开了音视频的用户列表中，本地在播放的用户列表。
 #discussion 断开重连、暂停恢复等操作不自动重置 `videoPlayingUsers`，除非对方用户掉线、离线等
 */
@property (nonatomic, readonly, copy, nullable) NSArray<BJLMediaUser *> *videoPlayingUsers;
```

```
// example：监听 videoPlayingUsers，videoPlayingUsers 表示当前正在播放的视频的对象，监听该属性的变化可以即时获取播放对象的最新信息
[self bjl_kvo:BJLMakeProperty(self.room.playingVM, videoPlayingUsers)
    observer:^BOOL(NSArray<BJLUser *> *value, NSArray<BJLUser *> *oldValue, BJLPropertyChange * _Nullable change) {
   NSLog(@"当前正在播放%td人的视频", value.count);
   return YES;
}];
```

3. 播放控制

通过音视频用户列表可以获取到用户的音视频状态信息，SDK 会自动播放 `BJLMediaUser` 的 `audioOn` 为 YES 的用户的声音，而是否播放视频画面是由用户是否在推视频流，即 `BJLMediaUser` 的 `videoOn` 状态，以及上层是否播放该用户的视频来决定的，默认不播放视频画面。

- 获取播放视图。

```
/**
 获取播放用户的视频视图
 #param userID 用户 ID
 #param mediaSource 视频源类型
 */
- (nullable __kindof BJLMediaUser *)playingUserWithID:(nullable NSString *)userID
                                                number:(nullable NSString *)userNumber
                                             mediaSource:(BJLMediaSource)mediaSource;

// example：获取老师主摄像头采集视频的视图
UIView *mainCameraView = [self.room.playingVM playingViewForUserWithID:self.room.onlineUsersVM
 mediaSource:BJLMediaSource_mainCamera];
```

- 打开 / 关闭 对象用户的视频。

```
/**
 设置播放用户的视频
 #param userID 用户 ID
 #param videoOn YES：打开视频，NO：关闭视频
 #param definitionIndex `BJLMediaUser` 的 `definitions` 属性的 index，参考 `BJLLiveDefinitionKey`、`BJLLiveDefinitionNameForKey()`
```

```
 #param mediaSource 视频源类型
 #return BJLError:
 BJLErrorCode_invalidArguments  错误参数，如`playingUsers`中不存在此用户；
 BJLErrorCode_invalidCalling   错误调用，如用户视频已经在播放、或用户没有开启摄像头。
 */
- (nullable BJLError *)updatePlayingUserWithID:(NSString *)userID
                                     videoOn:(BOOL)videoOn
                                  mediaSource:(BJLMediaSource)mediaSource;
- (nullable BJLError *)updatePlayingUserWithID:(NSString *)userID
                                     videoOn:(BOOL)videoOn
                                  mediaSource:(BJLMediaSource)mediaSource
                                 definitionIndex:(NSInteger)definitionIndex;
```

```
// example：关闭 user 对应的视频流，user 为`BJLMediaUser`实例
[self.room.playingVM updatePlayingUserWithID:user.ID
                                     videoOn:NO
                                  mediaSource:user.mediaSource];
```

```
// example：播放老师主摄像头采集的视频
BJLUser *onlineTeacher = self.room.onlineUsersVM.onlineTeacher;
if (onlineTeacher) {
    // 指定目标视频源类型为主摄像头采集
```

```
    BJLMediaSource targetMediaSource = BJLMediaSource_mainCamera;
    // 获取老师主摄像头的的视频视图
    UIView *mainCameraView = [self.room.playingVM playingViewForUserWithID:onlineTeacher.ID mediaSource:targetMediaSource];
    // 将播放视图添加到当前 viewController 的对应视图（布局自定）
    [self.playingView addSubview:mainCameraView];
    // 播放视频
    [self.room.playingVM updatePlayingUserWithID:onlineTeacher.ID
                                         videoOn:YES
                                     mediaSource:targetMediaSource];
}
```

- 自动播放视频并指定清晰度回调。

集成方可以通过 `BJLPlayingVM` 的 `autoPlayVideoBlock` 控制视频的自动播放，并指定视频的清晰度。集成方可以自定义一个视频黑名单，将用户主动关闭过的视频流加入黑名单中，当 `autoPlayVideoBlock` 回调时，根据回调的 `BJLMediaUser` 对应的视频流是否在黑名单内，决定是否自动播放。

```
/**
 自动播放视频并指定清晰度回调
 #discussion 传入参数 user 和 cachedDefinitionIndex 分别为 用户 和 上次播放该用户视频时使用的清晰度
 #discussion 返回结果 autoPlay 和 definitionIndex 分别为 是否自动播放视频 和 播放视频使用的视频清晰度
```

```
5. */
6. @property (nonatomic, copy, nullable) BJLAutoPlayVideo (^autoPlayVideoBlock)(BJLMediaUser *user, NSInteger cachedDefinitionIndex);
```

```
1. // example：主动关闭用户的视频流之后，将其加入黑名单，不再自动播放
2. [self.room.playingVM updatePlayingUserWithID:playingUser.ID videoOn:NO mediaSource:playingUser.mediaSource];
3. 
4. // videoKeyForUser:方法为用户的每一个视频源类型创建唯一的标识符，用于黑名单的存取
5. // 标识符建议使用 user.number-user.mediaSource 的字符串拼接形式，因为 ID、mediaID 会因为断网重连等情况发生改变
6. [self.autoPlayVideoBlacklist addObject:[self videoKeyForUser:playingUser] ?: @""];
7. 
8. // example：自动播放不在黑名单中的视频流
9. self.room.playingVM.autoPlayVideoBlock = ^BJLAutoPlayVideo(BOOL autoPlay, NSInteger definitionIndex)(BJLMediaUser *user, NSInteger cachedDefinitionIndex) {
10.     bjl_strongify(self);
11.     NSString *videoKey = [self videoKeyForUser:user];
12.     // 不在黑名单中的视频流，自动播放
13.     BOOL autoPlay = videoKey && ![self.autoPlayVideoBlacklist containsObject:videoKey];
14.     // 指定清晰度序号
```

```
    NSInteger definitionIndex = cachedDefinitionIndex;
    if (autoPlay) {
        NSInteger maxDefinitionIndex = MAX(0, (NSInteger)user.definitions.count - 1);
        definitionIndex = (cachedDefinitionIndex <= maxDefinitionIndex
                           ? cachedDefinitionIndex : maxDefinitionIndex);
    }
    return BJLAutoPlayVideoMake(autoPlay, definitionIndex);
};
```

- 停止播放。

```
// 停止播放
[self.room.playingVM updatePlayingUserWithID:user.ID videoOn:NO mediaSource:user.mediaSource];

// 移除该 user 的 playingView (playingView 获取方法参考播放视频部分)
[playingView removeFromSuperView];
```

- 获取播放视图的信息。

```
// 获取播放用户的清晰度
- (NSInteger)definitionIndexForUserWithID:(NSString *)userID
                                  mediaSource:(BJLMediaSource)mediaSource;

// 获取播放用户的视频视图宽高比
```

```
- (CGFloat)playingViewAspectRatioForUserWithID:(NSString *)userID
                                      mediaSource:(BJLMediaSource)mediaSource;
```

- 通过 `BJLPlayingVM` 设置所有播放视图的显示模式。如果需要为每一个视图单独设置不同的显示模式，可以设置成 fill 模式后，通过获取每个视图的实际比例，参考 `playingViewAspectRatioForUserWithID:mediaSource:` 方法获取，然后加入到实际想要显示的视图中，分别达到 fit 和 fill 的效果。

```
/** 播放画面显示模式，默认 BJLVideoContentMode_aspectFit */
@property (nonatomic) BJLVideoContentMode videoContentMode;

// example: 设置为 fill 的铺满模式
self.room.playingVM.videoContentMode = BJLVideoContentMode_aspectFill;
```

- 特别的针对拉CDN合流的教室，通过 `BJLPlayingVM` 可以设置是否要使用原画清晰度

```
/** 是否是原始清晰度，默认是原始清晰度 */
@property (nonatomic, readonly) BOOL originCDNVideoDefinition;

/**
 使用合流原始清晰度，默认原始清晰度
 #discussion 大班课合流模式下，参考 `playMixedVideo` 生效
 #discussion 伪直播、推流直播的场景下，参考 `BJLRoom.roomInfo.isMockLive` 和 `BJLRoom.roomInfo.isMockLive.isPushLive` 生效
```

```
 #param origin YES：原始清晰度 NO：使用低清晰度
 #return BJLError 不支持切换清晰度，参考 `BJLFeatureConfig.enableSwitchMixedVideoDefinitic
 */
- (nullable BJLError *)useOriginCDNVideoDefinition:(BOOL)origin;
```

4. 监听播放回调

播放回调信息通过监听 `self.room.playingVM` 的方法调用获取。

- 音视频用户列表覆盖更新。

列表覆盖更新目前主要有三种情况下触发：进入或者切换教室、断网重连、合流状态切换。

```
/**
 `playingUsers` 被覆盖更新
 #discussion 进教室后批量更新才调用，增量更新不调用
 #param playingUsers 音视频用户列表
 */
 [self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUsersDidOverwrite:extraPlayingUsers:)
         observer:^BOOL(NSArray * _Nullable now, NSArray * _Nullable old) {
     NSLog(@"playingUsersDidOverwrite");
     return YES;
 }];
```

- 用户开关音视频。

```
/**
```

```
用户开关音、视频
 #discussion - 某个用户主动开关自己的音视频、切换清晰度时发送此通知，但不包含意外掉线等情况
 #discussion - 正在播放的视频用户 关闭视频时 `videoPlayingUser` 将被设置为 nil、同时发送此通知
 #discussion - 进教室后批量更新 `playingUsers` 时『不』发送此通知
 #discussion - 音视频开关状态通过 `BJLMediaUser` 的 `audioOn`、`videoOn` 获得
 #discussion - definitionIndex 可能会发生变化，调用 `definitionIndexForUserWithID:` 可获取最新的取值
 #param now 新用户信息
 #param old 旧用户信息
 */
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidUpdate:old:)
        observer: (BJLMethodFilter)^BOOL(BJLMediaUser * _Nullable now, BJLMediaUser * _Nullable old) {
    bjl_strongify(self);
    if (now.isTeacher) {
        BOOL audioChanged = (now.audioOn != old.audioOn
                             && now.mediaSource == BJLMediaSource_mainCamera);
        BOOL videoChanged = (now.videoOn != old.videoOn);

        NSString *videoTitle = BJLVideoTitleWithMediaSource(now.mediaSource);
        if (audioChanged && videoChanged) {
            if (now.audioOn && now.videoOn) {
                [self showProgressHUDWithText: [NSString stringWithFormat:@"老师开启了麦克风
```

```
和%@", videoTitle]];
        }
        else if (now.audioOn) {
            [self showProgressHUDWithText:@"老师开启了麦克风"];
        }
        else if (now.videoOn) {
            [self showProgressHUDWithText:[NSString stringWithFormat:@"老师开启了%@", videoTitle]];
        }
        else {
            [self showProgressHUDWithText:[NSString stringWithFormat:@"老师关闭了麦克风和%@", videoTitle]];
        }
    }
    else if (audioChanged) {
        if (now.audioOn) {
            [self showProgressHUDWithText:@"老师开启了麦克风"];
        }
        else {
            [self showProgressHUDWithText:@"老师关闭了麦克风"];
        }
    }
    else { // videoChanged
        if (now.videoOn) {
            [self showProgressHUDWithText:[NSString stringWithFormat:@"老师开启了%@", videoTitle]];
        }
        else {
            [self showProgressHUDWithText:[NSString stringWithFormat:@"老师关闭了%@",
```

```
videoTitle]];
            }
        }
    }
    return YES;
}];
```

- 用户视频清晰度变化。

```
/**
 用户改变视频清晰度
 #param now 新用户信息
 #param old 旧用户信息
 #discussion 清晰度的变化可对比 `now` 与 `old` 的 `definitions` 属性得知
 */
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidUpdateVideoDefinitions:old:)
        observer:(BJLMethodFilter)^BOOL(BJLMediaUser * _Nullable now, BJLMediaUser * _Nullable old) {
NSLog(@"playingUserDidUpdateVideoDefinitions");
    return YES;
 }];
```

- 用户视频宽高比变化。

```
/**
 用户视频宽高比发生变化的通知
 #param videoAspectRatio 视频宽高比
 #param user             用户音视频流信息
 */
```

```
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingViewAspectRatioChanged:forUser:)
        observer:(BJLMethodObserver)^BOOL(CGFloat ratio, BJLMediaUser *user) {
    bjl_strongify(self);
    // 更新视频布局
    [self updateVideoViewConstranintsWithAspectRatio:ratio forUser:user];
    return YES;
}];
```

- 用户视频流加载状态。

可用于显示视频 loading 状态，建议直接监听 `BJLMediaUser` 的 `isLoading` 状态。

```
/**
 将要播放视频
 #discussion 播放视频的方法被成功调用时回调
 #param playingUser 将要播放的视频用户
 */
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidStartLoadingVideo:)
        observer:^BOOL(BJLUser *user) {
    bjl_strongify(self);
    [self tryToShowLoadingViewWithUser:user];
    return YES;
}];
```

```
/**
 视频播放成功
```

```
3. #discussion 用户视频播放成功
4. #param playingUser 播放的视频对应的用户
5. */
6. [self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidFinishLoadingVideo:)
7.         observer:^BOOL(BJLUser *user) {
8.     bjl_strongify(self);
9.     [self tryToCloseLoadingViewWithUser:user];
10.    return YES;
11. }];
```

5. 播放媒体文件

SDK 提供在直播间内播放媒体文件的功能，目前的实现上，移动端发起的播放媒体文件是教室内所有用户从 CDN 节点拉流播放的，不是直接推出的媒体流，播放时声音可以和其他用户的声音混合。

5.1 点播预热

暖场视频地址可以通过 `BJLRoomVM` 的 `getWarmingUpVideoListWithCompletion:` 方法获取，仅能在上课前获取到。

```
1. [self.room.roomVM getWarmingUpVideoListWithCompletion:^(BOOL success, BOOL isLoop, NSArray<NSString *> *_Nonnull videoList) {
2.     bjl_strongify(self);
3.     if (!success || !videoList) {
4.         return;
5.     }
6.     self.playerView = [self.room.playingVM vodPlayerViewWithURLString:URLString];
```

```
    [self insertSubview:self.playerView atIndex:0];
    [self.playerView bjl_remakeConstraints:^(BJLConstraintMaker *_Nonnull make) {
        make.edges.equalTo(self);
    }];
}];
```

大班课在教室上课之前可以播放点播预热视频，一般作为主讲人的视频显示。

```
/**
 通过 URL 获取视图
 #discussion 如果不是webrtc底层, 将会返回空值
 #discussion 获取视图后将会默认自动播放，从 0 开始播放，速率为 1.0
 #param videoURLString videoURLString
 */
- (nullable UIView *)vodPlayerViewWithURLString:(NSString *)videoURLString;
```

```
/** 播放点播预热视频 */
- (void)vodVideoPlay;

/** 暂停点播预热视频 */
- (void)vodVideoPause;

/**
 跳转播放视频的的时间
 #discussion 跳转的时间超过当前媒体文件的总时长时无效
 #param time 跳转到的目标时间，单位 秒
 */
```

```
- (void)seekVodPlayerToTime:(NSTimeInterval)time;

/** 停止播放点播暖场视频 */
- (void)stopVodPlayer;
```

5.2 云端媒体文件

专业小班课可以播放 `BJLDocumentVM` 中通过 `didLoadMediaFiles:` 回调教室中的媒体文件，媒体文件可以在教室内上传或者后台关联到教室中。播放的云端媒体文件一般显示成窗口视图。目前教室中仅支持同时播放一个媒体文件，如果存在多个，将会关闭之前播放的媒体文件。

```
/**
 通过媒体文件获取播放媒体文件视图
 #discussion 获取视图后将会默认自动播放，从 0 开始播放，速率为 1.0
 #param mediaFile BJLMediaFile
 */
- (nullable UIView *)cloudVideoViewWithMediaFile:(BJLMediaFile *)mediaFile;
```

```
/** 当前有其他用户正在播放媒体文件 */
@property (nonatomic, readonly) BOOL otherCloudVideoPlaying;
/**
 改变当前播放媒体文件的播放状态
 #discussion 当前没有正在播放的媒体文件时忽略
 #param play 是否播放
 */
- (void)updateCloudVideoPlayStatus:(BOOL)play;

```

```
/**
 跳转播放媒体文件的时间
 #discussion 跳转的时间超过当前媒体文件的总时长时无效
 #param time 跳转到的目标时间，单位 秒
 */
- (void)seekCloudVideoToTime:(NSTimeInterval)time;

/**
 改变当前播放媒体文件的播放速率
 #discussion 当前没有正在播放的媒体文件时忽略
 #param rate 播放速率，默认 1.0
 */
- (void)changeCloudVideRate:(CGFloat)rate;

/** 停止播放媒体文件 */
- (void)stopPlayCloudVideo;
```

6. 专业小班课 API

- 视频窗口位置更新。

专业小班课引入了同步的视频窗口，支持可以在各端统一比例的桌面上拖动用户的窗口，并且同步位置、吃力，可以通过相关的接口来实现这个功能。

```
/**
 专业版小班课 - 更新视频窗口
 #param mediaID 视频流标识
 #param action 更新类型，参考 BJLWindowsUpdateModel 的 BJLWindowsUpdateAction
 #param displayInfos 教室内所有视频窗口显示信息
 #return 调用错误, 参考 BJLErrorCode
```

```
 */
- (nullable BJLError *)updateVideoWindowWithMediaID:(NSString *)mediaID
                                             action:(NSString *)action
                                           displayInfos:(NSArray<BJLWindowDisplayInfo *> *)displayInfos;
```

```
/**
 专业版小班课 - 视频窗口更新通知
 #param updateModel 更新信息
 #param shouldReset 是否重置
 */
 - (BJLObservable)didUpdateVideoWindowWithModel:(BJLWindowUpdateModel *)updateModel
                                         shouldReset:(BOOL)shouldReset;


// example：根据通知更新窗口布局
[self bjl_observe:BJLMakeMethod(self.room.playingVM, didUpdateVideoWindowWithModel:shouldReset:)
        observer:(BJLMethodObserver)^BOOL(BJLWindowUpdateMod *updateModel, BOOL shouldReset) {
    bjl_strongify(self);
    if (shouldReset) {
        [self resetVideoWindowsWithModel:updateModel];
    }
    else {
```

```
        [self updateVideoWindowWithModel:updateModel];
    }
    return YES;
}];
```

- 老师/助教 将用户上下台。

百家云专业小班课引入了台上、台下用户的概念，台上的用户可以不需要举手就能开关自己的音视频，不管是否打开音视频，始终占用一路上麦并发数，用户一旦上台，就会保留在音视频用户列表中，可以通过相关接口控制用户的上下台。

```
// 用户上台
[self.room.playingVM requestAddActiveUser:user];
// 用户下台
[self.room.playingVM requestRemoveActiveUser:user];
```

```
// 用户上台成功回调
[self bjl_observe:BJLMakeMethod(self.room.playingVM, didAddActiveUser:)
        observer:^BOOL(BJLUser *user) {
    // bjl_strongify(self);
    NSLog(@"%@已上台", user.name);
    return YES;
}];

// 用户上台请求被服务端拒绝
[self bjl_observe:BJLMakeMethod(self.room.playingVM, didAddActiveUserDeny:responseCode:)
```

```
        observer:^BOOL(BJLUser *user, NSInteger responseCode) {
    //bjl_strongify(self);
    return YES;
}];

// 用户下台成功回调
[self bjl_observe:BJLMakeMethod(self.room.playingVM, didRemoveActiveUser:)
        observer:^BOOL(BJLUser *user) {
    // bjl_strongify(self);
    NSLog(@"%@已下台", user.name);
    return YES;
}];
```

举手发言

举手发言功能由 `BJLSpeakingRequestVM` 管理。

1. 学生举手发言

对于老师，只要进入教室成功并且处于上课状态，就会保持发言状态，可以随时向教室内的其他用户发布音、视频（进入教室成功通过监听到 `BJLRoom` 的 `enterRoomSuccess` 方法得知，上课状态则通过监听 `BJLRoomVM` 的 `liveStarted` 属性获取）。

对于学生，除了进入教室成功并且处于上课状态这两个条件之外，需要举手向老师发送申请，老师同意后才能进入发言状态。发送申请之前需要判断老师是否在教室以及当前是否处于上课状态，申请的处理结果可以通过监听获得，申请的超时时间固定为 30 秒，SDK 提供了相应的倒计时监听方法。

在百家云大班课中，举手是申请打开麦克风和摄像头的方式，在专业小班课中，对于台上的学生，举手仅是申请打开麦克风的方式，开关摄像头是不需要举手的，对于台下的学生，举手发言将会变成台上学生，并且打开麦克风。

- 判断当前是否禁止举手。

```
// 老师禁止学生举手状态
@property (nonatomic, readonly) BOOL forbidSpeakingRequest;
```

- 判断当前是否是发言状态，专业小班课不处理 speakingEnabled，专业小班课仅在打开音频时认为是发言状态。

```
/**
 学生: 发言状态
 #discussion 举手、邀请发言、远程开关音视频等事件会改变此状态
 #discussion 上层需要根据这个状态开启/关闭音视频，上层开关音视频前需要判断当前音视频状态
 #discussion 因为 `speakingDidRemoteControl:` 会直接开关音视频、然后再更新学生的 `speakingEnabled` */
@property (nonatomic, readonly) BOOL speakingEnabled;
```

- 举手申请发言。

```
/**
 学生: 发送发言申请
 #discussion 上课状态才能举手，参考 `roomVM.liveStarted`
 #discussion 发言申请被允许/拒绝时会收到通知 `speakingRequestDidReply:`
```

```
 #return BJLError:
 BJLErrorCode_invalidCalling    错误调用，如在非上课状态、或者禁止举手等情况下调用此方法；
 BJLErrorCode_invalidUserRole   错误权限，要求非试听学生权限。
*/
BJLError *error = [self.room.speakingRequestVM sendSpeakingRequest];
```

- 正在申请发言的学生，仅老师和助教可以取到。

```
// 正在申请发言的学生
@property (nonatomic, readonly, copy, nullable) NSArray<BJLUser *> *speakingRequestUsers;
```

- 监听举手发言申请的处理结果。

```
/**
 学生&老师: 发言申请被允许/拒绝
 #discussion 更新学生的 `speakingEnabled`
 #discussion 老师可以收到所有人发言状态的变更，比如学生自己取消、助教协助允许/拒绝
 #param speakingEnabled 发言申请是否被允许、关闭
 #param isUserCancelled 学生本人取消/请求超时自动取消
 #param user            申请发言的用户
 */
 - (BJLObservable)speakingRequestDidReplyEnabled:(BOOL)speakingEnabled
                                    isUserCancelled:(BOOL)isUserCancelled
                                               user:(BJLUser *)user;

```

```
exzample:

bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestDidReplyEnabled:isUserCancelled:
        observer:(BJLMethodObserver)^BOOL(BOOL speakingEnabled, BOOL isUserCancelled, BJLUser *user) {
    bjl_strongify(self);
    NSLog(@"发言申请已被%@", speakingEnabled ? @"允许" : @"拒绝");
    if (speakingEnabled) {
        //发言请求被批准，打开麦克风
        [self.room.recordingVM setRecordingAudio:YES
                                recordingVideo:NO];
        NSLog(@"麦克风已打开");
    }
    return YES;
}];
```

- 监听发言状态。

```
/**
 音视频被远程开启、关闭，导致发言状态变化
 #discussion 音视频有一个打开就开启发言、全部关闭就结束发言
 #discussion SDK 内部先开关音视频、然后再更更新学生的 `speakingEnabled` 的状态
 #discussion 参考 `BJLRecordingVM` 的 `recordingDidRemoteChangedRecordingAudio:recor
 #param enabled YES：开启，NO：关闭
```

```
 */
 - (BJLObservable)speakingDidRemoteControl:(BOOL)enabled;

example:
[self bjl_observe:BJLMakeMethod(self.room.speakingReq speakingDidRemoteControl:)
        observer:(BJLMethodObserver)^BOOL(BOOL enabled) {
    NSlog(@"发言状态被%@", enabled ? @"开启" : @"关闭");
    return YES;
}];
```

- 举手发言申请的自动取消倒计时。

```
/**
 举手自动取消倒计时总时长、更新间隔、剩余时间
 #discussion 调用 `sendSpeakingRequest` 举手时设置为 `speakingRequestTimeoutInterval` 秒
 #discussion 每 `speakingRequestCountdownStep` 秒更新，变为 0.0 表示举手超时，变为 - 1.0 表示计时被取消 */
@property (nonatomic, readonly) NSTimeInterval speakingRequestTimeoutInterval, speakingRequestCountdownStep, speakingRequestTimeRemaining;
```

```
// 监听发言申请的剩余持续时间：剩余时间 speakingRequestTimeRemaining 的值由SDK内部计时器控制，可通过监听该值的变化进行自定义的响应操作
[self bjl_kvo:BJLMakeProperty(self.room.speakingReques
```

```
speakingRequestTimeRemaining)
    options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld
    filter:^BOOL(NSNumber * _Nullable timeRemaining, NSNumber * _Nullable oldValue, BJLPropertyChange * _Nullable change) {
  return timeRemaining.doubleValue != oldValue.doubleValue;
}
    observer:^BOOL(NSNumber * _Nullable timeRemaining, NSNumber * _Nullable oldValue, BJLPropertyChange * _Nullable change) {
  NSLog(@"timeRemaining:%f/%f", timeRemaining, BJLSpeakingRequestTimeoutInterval);
  return YES;
}];
```

- 学生取消发言申请：**取消申请不会自动关闭音视频采集**，调用以下取消申请的方法之后 `BJLRecordingVM` 的 `speakingEnabled` 会变为 `NO`，可以事先监听该属性的变化，在监听的回调里调用 `[self.room.recordingVM setRecordingAudio:NO recordingVideo:NO]` 关闭音视频采集，完全结束发言。

```
[self.room.speakingRequestVM stopSpeakingRequest];
```

- 停止发言：正在发言的用户，将音视频采集全部关闭则会自动关闭发言状态。

```
[self.room.recordingVM setRecordingAudio:NO recordingVideo:NO];
```

2. 学生处理发言邀请

学生还可以收到老师的发言邀请，接受之后将进入发言状态。

老师邀请发言需要学生同意，强制发言不需要学生同意，学生将会直接变成发言状态。

- 老师、助教邀请/强制发言。

```
/**
 老师: 邀请学生发言
 #param invite  YES 邀请、NO 取消邀请
 */
- (nullable BJLError *)sendSpeakingInviteToUserID:(NSString *)userID invite:(BOOL)invite;
```

- 监听收到的发言邀请：监听 `BJLSpeakingRequestVM` 的 `didReceiveSpeakingInvite:` 方法，`invite` 参数为 YES 时表示收到邀请，为 NO 时表示邀请被取消。

```
[self bjl_observe:BJLMakeMethod(self.room.speakingRequest didReceiveSpeakingInvite:)
        observer:(BJLMethodObserver)^BOOL(BOOL invite) {
    if (invite) {
        NSLog(@"received speaking invitaion");
    }
    else {
        NSLog(@"speaking invitation canceled");
    }
    return YES;
}];
```

- 接受或拒绝发言邀请。

```
[self.room.speakingRequestVM responseSpeakingInvite:YES]; // YES：接受，NO：拒绝
```

3. 老师处理发言申请

- 监听正在申请发言的学生列表：列表数组 `speakingRequestUsers` 在 SDK 内部即时更新，监听它的变化可以添加一些自定义的后续操作。

```
[self bjl_kvo:BJLMakeProperty(self.room.speakingRequestVM.speakingRequestUsers)
    options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld
    observer:^BOOL(NSArray<BJLUser *> * _Nullable value, NSArray<BJLUser *> * _Nullable oldValue, BJLPropertyChange * _Nullable change) {
    NSLog(@"value: %lu elements, oldValue: %lu elements", (unsigned long)value.count, (unsigned long)oldValue.count);
    return YES;
}];
```

- 允许 / 拒绝发言。

```
[self.room.speakingRequestVM replySpeakingRequestToUserID:user.ID allowed:YES]; //YES：允许，NO：拒绝
```

- 监听收到发言申请的通知：发送申请的 user 将被自动添加到 `speakingRequestUsers` 中，这里可添加自定义的后续操作。

```
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestVM.
didReceiveSpeakingRequestFromUser:)
        observer:^BOOL(BJLUser *user) {
    bjl_strongify(self);
    // 自定义后续操作，以同意申请为例:
    [self.room.speakingRequestVM replySpeakingRequestToUserID:user.ID allowed:YES];
    NSLog(@"%@ 请求发言、已同意", user.name);
    return YES;
}];
```

- 远程开关学生音、视频。

```
/**
 老师: 远程开关学生音、视频
 #param user 对象用户，不能是老师
 #param audioOn YES：打开音频采集，NO：关闭音频采集
 #param videoOn YES：打开视频采集，NO：关闭视频采集
 #discussion 打开音频、视频会导致对方发言状态开启
 #discussion 同时关闭音频、视频会导致对方发言状态终止
 @see `speakingRequestVM.speakingEnabled`
 #return BJLError:
 BJLErrorCode_invalidArguments  错误参数；
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限。
```

```
 */
BJLError *error = [self.room.recordingVM remoteChangeRecordingWithUser:user audioOn:NO videoOn:NO];
```

直播间音视频设置

SDK 对前后台切换、麦克风占用等情况进行了处理，教室内音视频效果可以通过 `BJLMediaVM` 控制。

1. 是否支持后台音频播放

```
/** 是否支持后台音频, 默认支持 */
@property (nonatomic, readonly) BOOL supportBackgroundAudio;
- (BJLError *)updateSupportBackgroundAudio:(BOOL)supportBackgroundAudio;
```

2. 是否支持后台音频采集

```

/** 是否支持后台采集声音, 默认不支持 */
@property (nonatomic, readonly) BOOL supportBackgroundRecordingAudio;
- (BJLError *)updateSupportBackgroundRecordingAudio:(BOOL)supportBackgroundRecordingAudio;
```

3. 是否播放视频静音

```
/** 是否播放视频静音, 默认不静音 */
```

```
@property (nonatomic, readonly) BOOL needMutePlayingAudio;
- (BJLError *)updateNeedMutePlayingAudio:(BOOL)needMutePlayingAudio;
```

4. WebRTC 教室回调

对于 `WebRTC` 教室（通过 `self.room.featureConfig.isWebRTC` 判断），如果不是播放合流时，`BJLMediaVM` 的 `inLiveChannel` 属性表示是否进入直播频道，这是使用音视频功能的前提。

```
// WebRTC：是否已进入直播频道
@property (nonatomic, readonly) BOOL inLiveChannel;

// webRTC 进入直播频道失败
[self bjl_observe:BJLMakeMethod(self.room.mediaVM, enterLiveChannelFailed)
        observer:^BOOL{
    // bjl_strongify(self);
    NSLog(@"进入直播频道失败，将无法使用音视频");
    return YES;
}];

// webRTC 直播频道断开提示
[self bjl_observe:BJLMakeMethod(self.room.mediaVM, didLiveChannelDisconnectWithError:)
        observer:^BOOL(NSError *error){
    // bjl_strongify(self);
    NSLog(@"直播频道已断开，请退出重试");
    return YES;
}];
```

```
19.
20. // 音视频网络状态更新回调
21. [self
bjl_observe:BJLMakeMethod(self.room.mediaVM,
mediaNetworkStatusDidUpdateWithUser:status:)
22.         observer:
(BJLMethodObserver)^BOOL(BJLMediaUser *user,
BJLMediaNetworkStatus status) {
23.     bjl_strongify(self);
24.     // 处理网络状态变化
25.     [self updateMediaNetworkStatus:status
forUser:user];
26.     return YES;
27. }];
28.
29. // 音视频丢包率更新回调
30. [self
bjl_observe:BJLMakeMethod(self.room.mediaVM,
mediaLossRateDidUpdateWithUser:videoLossRate:a
31.         observer:
(BJLMethodObserver)^BOOL(BJLMediaUser *user,
CGFloat videoLossRate, CGFloat audioLossRate){
32.     bjl_strongify(self);
33.     // 处理丢包率变化
34.     [self updateVideoLossRate:videoLossRate
35.                 audioLossRate:audioLossRate
36.                     forUser:user];
37.     return YES;
38. }];
39.
40. // 音量更新回调
41. [self
bjl_observe:BJLMakeMethod(self.room.mediaVM,
volumeDidUpdateWithUser:volume:)
42.         observer:
(BJLMethodObserver)^BOOL(BJLMediaUser *user,
```

```
CGFloat volume){
    bjl_strongify(self);
    // 处理音量变化
    [self updateVolume:volume forUser:user];
    return YES;
}];
```

下载为pdf格式