iOS 直播 Core SDK 2.x

变更记录：**https://git2.baijiashilian.com/open-ios/BaijiaYun/-/blob/master/ChangeLog/BJLiveCore-CHANGELOG.md**

git 链接：**https://git2.baijiashilian.com/open-ios/BaijiaYun**

App 下载：**https://itunes.apple.com/app/id1146697098?ls=1&mt=8**

旧版 SDK 文档：**iOS 直播 Core SDK 1.x**

SDK 升级文档：**iOS 1.x > 2.x SDK 升级整体说明**

功能介绍

百家云 iOS 直播 Core SDK 提供直播间场景及相应的一系列直播功能, 包括音视频推拉流、信令服务器通信、聊天服务器通信等，不包含 UI 资源，提供的 demo 可以较为完整的体验各个功能模块。包含 UI 的 SDK 请参考 iOS 直播 UI SDK, UI SDK 提供了一个针对教育场景下师生互动模板，包含一套完整的直播间 UI，集成工作量小，便于快速开发。 **集成 2.2.0 或以上版本的 SDK 要求 Xcode 的版本至少为 11.0，集成 2.10.4 或者以上版本的 SDK 要求 Xcode 的版本至少为 12.1**。

1. 概念

直播间主要内容	描述
老师	主讲人，拥有直播间最高权限，可以 设置上下课、发公告、处理他人举手、 远程开关他人麦克风和摄像头、 开关录课、开关聊天禁言
助教	管理员，拥有部分老师的权限， 老师可以对助教的权限进行管理
学生	听讲人，权限受限， 无法对他人的直播间内容进行管理
教室	直播间，提供创建、 管理等一系列功能。提供上课、 下课等接口， 大多数功能模块只有在上课状态下有效
举手	学生申请发言， 老师和管理员可以允许或拒绝
发言	发布音频、视频，SDK 层面发言不要求举手状态
采集	通过设备摄像头、 麦克风获取自己的本地视频、音频数据
播放	播放他人发布的视频， 支持同时播放多个人的视频
录课	云端录制课程，可以生成教室回放观看
聊天	直播间内的群聊、私聊功能， 支持发送图片、表情
课件	课件第一页是白板， 主要用于添加画笔；老师可上传文件、 图片格式的课件， 上传成功之后可在直播间内显示；支持 PPT 动画

画笔	老师、助教或发言状态的学生可以在 白板和 PPT 上添加、清除画笔； 添加画笔的用户当前的 PPT 页必须与老师保持一致
公告	由老师编辑、发布，可包含跳转链接， 即时更新
测验	学生收到老师发布的测验、进行答题
跑马灯	教室内文字浮层， 用于品牌标志或者个性化定制信息
课后评价	课程结束时自定义对课程， 主讲人的评价问卷反馈
点名	教室内老师发起点名，用于签到等情况
专注度	判断学生是否在教室内上课， 可以对不专注学生发出提醒
点赞	奖励学生良好的课堂表现
红包雨	抢红包，学分等课堂互动
问答	学生提问，助教或老师回答， 发布后全员可见
学情报告	包括通过 AI 人脸识别， 截取出特征表情画面的等教室学习情况 的统计报告
计时器	给定时长，计时进行互动
抢答器	发布抢答问题，与抢答学生互动
答题器	较测验更轻量的问答互动功能
抽奖	标准多人抽奖和口令抽奖
云摄	外接移动端设备作为摄像头，

	可以用于老师在教学过程中从不同角度显示老师的视频画面
自习室	在线连麦自习、辅导

2.主要功能

1.教室管理

功能	对应文件
进入 / 退出教室及相应的事件监听	BJLRoom.h
断开重连	BJLRoomVM.h
大小班切换	BJLRoomVM.h
进入教室的加载状态监听	BJLLoadingVM.h
老师或助教：上课 / 下课	BJLRoomVM.h
定制信令	BJLRoomVM.h

2.在线用户信息管理

功能	对应文件
加载在线用户信息	BJLOnlineUsersVM.h
监听用户进入、退出教室	-
主讲人切换	-
踢出教室用户管理	-

3.音视频采集

功能	对应文件

开启 / 关闭音视频采集	BJLRecordingVM.h
音视频采集状态监听	-
外接移动端设备作为摄像头	-
采集设置：视频方向，清晰度， 美颜，前后摄像头	-
禁止、允许开启麦克风	-
音频源数据回调	-

4. 视频播放

功能	对应文件
播放、关闭指定用户的视频	BJLPlayingVM.h
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， 视图显示模式，清晰度，水印等	-

5. 音视频设置

功能	对应文件
设置用于音视频的 上行 / 下行 链路的类型：UDP / TCP	BJLMediaVM.h
后台播放、采集，全局开启、 解除静音，网络状态、丢包率信息	-

6. 举手、发言邀请

功能	对应文件

学生举手、取消举手， 老师处理举手申请	BJLSpeakingRequestVM.h
学生接收、处理发言邀请	-
禁止、允许举手	-

7. 课件管理

功能	对应文件
上传、添加课件，删除课件， 课件翻页	BJLDocumentVM.h
加载课件	-
课件操作授权	-
小黑板管理	-

8. 画笔

功能	对应文件
设置画笔类型，画笔操作模式， 颜色，线宽等	BJLDrawingVM.h
文档画笔，小黑板画笔授权、开启、 关闭	-
激光笔显示、移动	-
添加图片画笔	-

9. 聊天

功能	对应文件
群聊、私聊发送消息（文字、图片、	BJLChatVM.h

表情）	
监听收到消息	-
禁言	-
置顶消息	-
聊天翻译	-

10. 录课

功能	对应文件
老师或助教： 监听云端录课不可用的通知，获取云端录课状态，开启/停止云端录课	BJLServerRecordingVM.h
录课结束请求生成回放	-

11. 教室工具

功能	对应文件
进入教室时间，上课时间，排课时间	BJLRoomVM.h
助教上麦、画笔、文档、公告、 上下课、禁言、踢人、 云端录制权限管理	-
发布，获取教室公告，监听公告变化	-
跑马灯内容，样式	-
课后评价	-
点名时长，收到点名，答到	-
专注度检测，不专注提醒	-

点赞发送、收取，点赞统计	-
红包雨创建、发布、收取、抢红包， 获取抢红包结果，红包排行榜	-
测验新增、删除、加载、发布、结束、 提交，获取测验列表	-
问答创建，发布，取消发布、回复， 禁止、允许提问，加载问答数据	-
网页同步打开、关闭	-
课后学情生成，获取报告数据	-
计时器发布、暂停、结束	-
抢答器发布、收到、抢答、关闭、撤销	-
答题器发布、结束、提交、关闭、 撤销，获取用户答题数据	-
参与标准抽奖和口令抽奖， 中奖后提交联系方式	-

12. 自习室

功能	对应文件
老师或助教：切换自习模式	BJLStudyRoomVM.h
获取自习室公告、时长排行榜、 学生列表	-

3. 错误码参考 NSError+BJLError.h

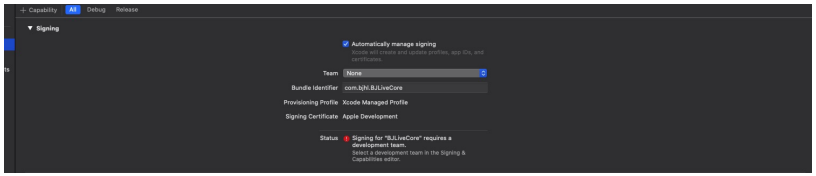
Demo

1. Demo 源文件

demo 源文件在 git 中。

2. Demo 编译、运行

- 在 demo 的工程目录下执行 `pod update` 。
- 使用 Xcode 打开 demo 文件夹下的 `BJLiveCore.xcworkspace` 文件。
- 选择运行设备：**模拟器运行 demo 时无法采集音视频**；真机运行时，需要设置好 `development team` ：



- 使用 Xcode 运行 demo。

3. Demo 体验

- demo 运行成功后将进入如下登录界面，需要输入机构代码（参考 专属域名说明），六位参加码及用户名才能进入教室。其中参加码通过使用 百家云后台 或者 API 创建一个教室获得，如果是八位参加码，需要取到其中一个可进入教室的六位参加码，用户名可自定义。

- 教室加载成功之后进入如下主界面，包含课件、采集、播放、控制台（显示教室动态及聊天消息）等部分，参考红色标注。


Carrier
8:45 PM
发送
聊天消息输入框
采集
采集视图：用于显示本地摄像头所采集的视频内容
课件视图：用于显示白板、课件、画笔等内容
播放1
播放2
播放视图：用于显示其他发言用户发布的视频内容
播放3
播放4
liveStarted: YES
收到历史测验: (
) - 正确 0, 错误 0
chat Teacher : Hello
控制台视图，用于显示当前教室的动态和聊天消息
Carrier
10:44 AM
发送
采集
播放1
播放2
采集
全部打开
打开麦克风
打开摄像头
UDP > TCP

引入 SDK

SDK 支持 **iOS 9.0 及以上** 的系统，**iPhone**、**iPad** 等设备，**集成 2.0.0 或以上版本的 SDK 要求 Xcode 的版本至少为 9.0**，由于 SDK 依赖关系复杂、手动配置繁琐，建议使用 CocoaPods 方式引入。

- Podfile 中设置 source：

```
source 'https://github.com/CocoaPods/Specs.git'
source 'https://git2.baijiashilian.com/open-ios/specs.git'
```

- Podfile 中引入 BJLiveCore，**SDK 对于集成进行了调整，如果不需要使用 AVSDK，直接引入 SDK 即可，如果需要使用 AVSDK，仍然需要将引入 SDK 之后的语句加入。特别的，点播和回放 SDK 的脚本语句不需要写入了，可以直接移除。**

- 不集成 AVSDK 写法：

```
pod 'BaijiaYun/BJLiveCore', '~> 2.11.0'
```

- 集成 AVSDK 写法如下：

```
pod 'BaijiaYun/BJLiveCore', '~> 2.11.0'

# 用于动态引入 Framework，避免冲突问题
script_phase \
:name => '[BJLiveCore] Embed Frameworks',
```

```
:script =>
'Pods/BaijiaYun/frameworks/EmbedFrameworks.sh',
:execution_position => :after_compile

# 用于清理动态引入的 Framework 用不到的架构，避免发布 AppStore 时发生错误，需要写在动态引入 Framework 的 script 之后
script_phase \
:name => '[BJLiveBase] Clear Archs From Frameworks',
:script =>
'Pods/BaijiaYun/frameworks/ClearArchsFromFramew
 "BJHLMediaPlayer.framework"',
:execution_position => :after_compile
```

- 工程目录下执行 `pod install`，初次集成需要执行 `pod update` 更新 CocoaPods 的索引。

版本升级

版本号格式为 `大版本.中版本.小版本[-alpha(测试版本)/beta(预览版本)]`：

- 测试版本和预览版本可能不稳定，请勿随意尝试。
- 小版本升级只改 BUG、UI 样式优化，不会影响功能。
- 中版本升级、修改功能，更新 UI 风格、布局，会新增 API、标记 API 即将废弃，但不会导致现有 API 不可用。
- 大版本任何变化都是有可能的。

首次集成建议选择最新正式版本(版本号中不带有 `alpha`、`beta` 字样)，**版本升级后请仔细阅读 ChangeLog**，指定版本的方式有一下几种：

- 固执型：`pod update` 时不会做任何升级，但可能无法享受到最新的 BUG 修复，建议用于 0.x 版本。

```
pod 'BaijiaYun/BJLiveCore', '2.0.0'
```

- 稳妥型**(推荐)**：`pod update` 时只会升级到更稳定的小版本，而不会升级中版本和大版本，不会影响功能和产品特性，升级后需要 **适当测试**。

```
pod 'BaijiaYun/BJLiveCore', '~> 2.0.0'
```

- 积极型：`pod update` 时会升级中版本，但不会升级大版本，及时优化，但不会导致编译出错不可用，升级后需要 **全面测试**。

```
pod 'BaijiaYun/BJLiveCore', '~> 2.0'
```

- 激进型**(不推荐)**：`pod update` 时会升级大版本，可能导致编译出错、必须调整代码，升级后需要 **严格测试**。

```
pod 'BaijiaYun/BJLiveCore'
```

工程设置

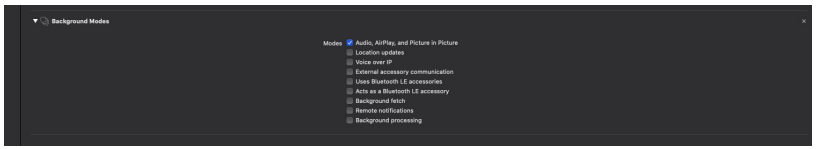
- 隐私权限：在 `Info.plist` 中添加麦克风、摄像头、相册访问描述。

```
Privacy - Microphone Usage Description 用于语音上课、发言
Privacy - Camera Usage Description 用于视频上课、发言，拍照传课件、聊天发图
Privacy - Photo Library Usage Description 用于上传课件、聊天发图
```

Privacy - Camera Usage Description	String	视频
Privacy - Microphone Usage Desc...	String	音频
Privacy - Photo Library Usage Des...	String	课件

```
<key>NSMicrophoneUsageDescription</key>
<string>用于语音上课、发言</string>
<key>NSCameraUsageDescription</key>
<string>用于视频上课、发言，拍照传课件、聊天发图</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>用于上传课件、聊天发图</string>
```

- 后台任务（**打开这一选项之后，在 App 提交审核时，强烈建议录制一个视频，说明 App 确实用到了后台播放，否则审核很有可能不通过**）：在 `Project > Target > Capabilities` 中打开 `Background Modes` 开关，选中 `Audio, Airplay, and Picture in Picture`。



Hello World

可参考 demo 中的 BJRoomViewController。

1. 要点说明

1.1 流程概述

SDK 所有的功能需要在 `教室` 中完成，**进入教室成功之后才能正常使用各个功能模块**。进入教室需要创建一个 `BJLRoom` 的实例，然后调用它的相关方法实现业务逻辑。进入教室成功之

后， 可通过 `BJLRoom` 中定义的各种 `ViewModel` 管理相应的功能模块，对应关系参考 主要功能。流程要点总结如下：

- `BJLRoom` 是直播功能的入口，用于创建、进入、退出教室。
- 教室内各个功能通过对应的 `ViewModel`（以下简称 VM）来管理。
- 所有 VM 及其所有属性支持 `KVO` 以便监听状态变化（除非额外注释说明），返回类型值为 `BJLObservable` 的方法表示可监听、用于监听事件。
- **VM 在创建教室时被初始化，此时可以添加监听，但 VM 的属性没有与服务端同步，也不能调用 VM 的方法与服务端交互。**
- 调用 `enter` 方法开始发起网络请求、进教室，可以通过 `room.loadingVM` 获取加载进度、成功和失败等，成功或失败后 `room.loadingVM` 被设置为空。
- **`room.state` 是教室的在线状态，进入教室成功状态为 `BJLRoomState_connected`，此时 VM 的属性与服务端完成同步，同时可调用 VM 方法与服务端交互。**

1.2 属性、方法监听方式：Block

我们使用 `NSObject+BJLObserving.h` 中的 Block 监听方式监听属性变化和方法，相比 `RAC` 更简单、高效、方便调试；self 和被监听对象 dealloc 时都会自动取消监听，相比普通 KVO 更简单、易用、且安全。

1.2.1 通过 Block 方式进行 KVO

- KVO 调用方式，添加 KVO 监听时的对象不能为空，必须在对象存在时才能对对象的属性就行监听，block 接收的返回值必须声明为对象，数值类型的接收值使用 NSValue 或者泛型，支持 filter - 可选：

```
// example: 监听教室上课状态
bjl_weakify(self);
```

```
[self bjl_kvo:BJLMakeProperty(self.room.roomVM,
// 对象，不能为 nil
                              liveStarted) // 属性名，支持代码
自动完成
      filter:^BOOL(NSNumber *value, NSNumber
*oldValue, BJLPropertyChange * _Nullable
change) { // 过滤
          return oldValue.boolValue !=
value.boolValue;  // 返回 NO 丢弃
      }
    observer:^BOOL(NSNumber *value,
NSNumber *oldValue, BJLPropertyChange *
_Nullable change) { // 处理
          bjl_strongify(self);
          // console 为自定义的控制台视图
          [self.console printFormat:@"liveStarted:
%@", NSStringFromBOOL(value.boolValue)];
          return YES; // 返回 NO 取消 KVO
    }];
```

- 支持两种方式取消某次 KVO，并且 self 或被监听对象 dealloc 时都会自动取消监听。

```
// example: 监听教室上课状态
bjl_weakify(self);
id<BJLObservation> observation =
   [self
bjl_kvo:BJLMakeProperty(self.room.roomVM,
liveStarted)
        observer:^BOOL(NSNumber *value,
NSNumber *oldValue, BJLPropertyChange *
_Nullable change) {
          bjl_strongify(self);
          [self.console printFormat:@"liveStarted:
%@", NSStringFromBOOL(value.boolValue)];
```

```
        return YES; // 1. 返回 NO 取消 KVO
    }];
[observation stopObserving]; // 2. 取消 KVO
```

1.2.2 通过 Block 方式监听方法调用

- 监听方法调用，参数要求是支持 KVO 的方法，如果方法的参数是数值类型，可以使用 NSValue 或者泛型的方式接收返回值，或者在 block 返回值前添加 (BJLMethodObserver)，支持 filter - 可选：

```
// example: 监听 '即将退出教室' 事件
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, // 对象，不能为 nil
                                roomWillExitWithError:) // 方法
          filter:(BJLMethodFilter)^BOOL(BJLError *error) { // 过滤
              return !!error; // 返回 NO 丢弃
          }];
        observer:(BJLMethodObserver)^BOOL(BJLError *error) { // 处理
            bjl_strongify(self);
            [self.console printFormat:@"roomWillExitWithError: %@", error];
            return YES; // 返回 NO 取消监听
        }];
```

- 支持 **多个参数**：

```
// example: 监听发言用户的状态变化
bjl_weakify(self);
```

```
[self
bjl_observe:BJLMakeMethod(self.room.playingVM,
playingUserDidUpdate:old:)
        observer:^BOOL(BJLUser *now,
                       BJLUser *old) {
            bjl_strongify(self);
            [self.console
printFormat:@"playingUserDidUpdate:old: %@,
%@", now, old];
        }];
```

- 支持两种方式取消某次监听，并且 self 或被监听对象 dealloc 时都会自动取消监听。

```
// example: 监听 '即将退出教室' 事件
bjl_weakify(self);
id<BJLObservation> observation =
   [self bjl_observe:BJLMakeMethod(self.room,
roomWillExitWithError:)
        observer:
(BJLMethodObserver)^BOOL(BJLError *error) {
            bjl_strongify(self);
            [self.console
printFormat:@"roomWillExitWithError: %@",
error];
            return YES; // 1. 返回 NO 取消监听
        }];
[observation stopObserving]; // 2. 取消监听
```

1.3 新版小班课支持的设备

教室类型为新版小班课（参考 BJLRoomType）的教室，只支持 64-bit 设备进入教室，32-bit 设备进教室时通过 `enterRoomFailureWithError:` 返回错误码

`BJLErrorCode_enterRoom_unsupportedDevice`，参考 iOS Device Summary。

- iPad: 1、2、3、4、mini 1 是 32-bit，其它都是 64-bit。
- iPhone: 5、5C 之前的设备是 32-bit，5S 开始是 64-bit。
- iPod Touch: 1、2、3、4、5 是 32-bit，6 开始是 64-bit。

2. 引入头文件

```
1. #import <BJLiveCore/BJLiveCore.h>
```

3. 创建、进入教室

创建、进入教室的整体流程如下：

- 设置专属域名前缀。
- 在自己定义的相关文件中定义一个 `BJLRoom` 的属性 `room`，用于管理教室。
- 使用教室相关信息将 `room` 属性实例化。
- 为教室的加载、进入、退出等事件添加监听和相应的回调处理。其中对加载任务的监听可以获取进教室的加载过程中每一个步骤的执行状态和出错时的错误信息，便于调试，也可以用来展示加载进程；对进入、退出的监听获取出现异常时的 error 信息。回调处理可以根据自身需求进行自定义，为教室管理做好准备。
- 添加断开重连处理。**如果不添加断开重连的回调处理，SDK 会默认在断开时自动重连**，重连过程中遇到错误将退出教室、抛出异常。
- 调用 `BJLRoom` 定义的 `enter` 方法进入教室，监听到进入成功之后，身份为老师的用户可以发送上课通知。

3.1 设置专属域名前缀

- BJLiveCore SDK 1.3.5 及之后版本支持设置专属域名。

- 设置专属域名前缀，需要在创建 BJLRoom 实例之前设置。例如专属域名为 demo123.at.baijiayun.com，则前缀为 demo123，参考 专属域名说明。

```
[BJLRoom setPrivateDomainPrefix:@"yourDomainPrefix"];
```

3.2 定义教室属性

```
@property (nonatomic) BJLRoom *room;
```

3.3 创建教室：可通过教室 ID 或参加码两种方式进行

- 教室 ID 方式：教室ID通过使用 百家云后台 或者 API 创建一个教室获得；签名参数通过 签名参数 sign 计算方法 获得。

```
/**
教室ID方式
@param userNumber    用户编号，合作方账号体系下的用户ID号，必须是数字
@param userName      用户名
@param userAvatar    用户头像 URL(nullable)
@param userRole      用户角色:老师、学生等
@param roomID        教室 ID
@param groupID       分组 ID, 不分组传0
@param apiSign       签名
*/

// 创建用户实例
BJLUser *user = [self userWithNumber:number
                                name:name
                             groupID:groupID
                              avatar:avatar
```

```
                                role:role];
// 创建教室
self.room = [BJLRoom roomWithID:roomID
                        apiSign:apiSign
                           user:user];
```

- 参加码方式：参加码同样通过使用 百家云后台 或者 API 创建一个教室获得。

```
/**
参加码方式
@param roomSecret    教室参加码
@param userName      用户名
@param userAvatar    用户头像 URL，可传空值
*/
self.room = [BJLRoom
roomWithSecret:roomSecret
                          userName:userName
                        userAvatar:nil];
```

3.4 进出教室

- 监听进入、退出教室等事件。

```
// 监听进入教室成功
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room,
enterRoomSuccess)
        observer:^BOOL() {
            bjl_strongify(self);
            if (self.room.loginUser.isTeacher) {
                // 身份为老师，通知学生上课
                [self.room.roomVM
sendLiveStarted:YES];
            }
```

```
        else {
            // 身份非老师，监听老师上课状态变化
            [self bjl_kvo:BJLMakeProperty(self.room.roomVM, liveStarted)
                    filter:^BOOL(NSNumber *value, NSNumber *oldValue, BJLPropertyChange * _Nullable change) {
                        return oldValue.boolValue != value.boolValue;
                    }
                  observer:^(, NSNumber *valueNSNumber *oldValue, BJLPropertyChange * _Nullable change) {
                    // 上课状态发生变化后的响应操作
                    // bjl_strongify(self);
                    NSLog(@"liveStarted: %@", value.boolValue? @"YES" : @"NO");
                }];
        }
        // 处理进教室后的逻辑
        [self didEnterRoom];
        return YES;
    }];
```

```
// 监听进入教室失败
[self bjl_observe:BJLMakeMethod(self.room, enterRoomFailureWithError:)
      observer:^BOOL(BJLError *error) {
        NSLog(@"进入教室失败:%@", error);
        return YES;
    }];
```

```
// 监听准备退出教室，error 为 nil 表示主动退出
```

```
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room,
roomWillExitWithError:)
        observer:^BOOL(BJLError *error) {
            bjl_strongify(self);
            if (self.room.loginUser.isTeacher) {
                // 通知学生下课
                [self.room.roomVM
sendLiveStarted:NO];
            }
            return YES;
        }];
```

```
// 监听退出教室，error 为 nil 表示主动退出
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room,
roomDidExitWithError:)
        observer:^BOOL(BJLError *error) {
            bjl_strongify(self);
            if (error) {
                // 获取错误信息
                NSString *message = error ? [NSString
stringWithFormat:@"%@ - %@",

error.localizedDescription,

error.localizedFailureReason] : @"错误";
                NSLog(@"error: %@", message);
            }
            // 自定义的退教室处理
            [self exit];
            return YES;
        }];
```

- 进入、退出教室。

```
// 进入教室
[self.room enter];

// 退出教室
[self.room exit];
```

3.5 断线重连

```
bjl_weakify(self);
[self.room setReloadingBlock:^(BJLLoadingVM * _Nonnull reloadingVM, void (^ _Nonnull callback)(BOOL)) {
    bjl_strongify(self);
    [self showAlertWithTitle:@"加载失败"
                     message:@"是否重连?"
                reloadCallback:^{
                    NSLog(@"网络连接断开，正在重连...");
                    // 自定义方法，处理重连的加载任务，可参考 demo
                    [self makeEventsForLoadingVM:reloadingVM];
                    NSLog(@"网络连接断开：重连");
                    callback(YES);
                }
                cancelCallback:^{
                    callback(NO);
                }];
  }];
```

3.6 上下课

```
// 上课
```

```
BJLError *error = [self.room.roomVM sendLiveStarted:YES];

// 下课
BJLError *error = [self.room.roomVM sendLiveStarted:NO];
```

3.7 准备进入教室：添加状态监听

- 监听加载任务：加载任务的相关属性及方法包含于 `self.room.loadingVM` 中。

```
// 监听进入教室的加载任务的变化
bjl_weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room, loadingVM)
      filter:^BOOL(id value, id oldValue, BJLPropertyChange * _Nullable change) {
          return !!value;
      }
    observer:^BOOL(BJLLoadingVM *value, id oldValue, BJLPropertyChange * _Nullable change) {
        bjl_strongify(self);
        // 自定义方法，处理当前的加载任务，可参考 demo
        [self makeEventsForLoadingVM:value];
        return YES;
    }];
```

```
// 监听加载进度
[self bjl_observe:BJLMakeMethod(self.room.loadingVM, loadingUpdateProgress:)
```

```
        observer:(BJLMethodObserver)^BOOL(CGFloat progress) {
            NSLog(@"loading progress: %f", progress);
            return YES;
        }];
```

```
/** 加载任务每一步骤停止时的回调
 @param step            当前加载步骤
 @param reason          停止原因
 @param error           具体错误
 @param continueCallback  回调：continueCallback(NO): 取消加载;
                                continueCallback(YES): 错误可忽略? 继续下一步骤 : 重试当前步骤
*/
bjl_weakify(self);
// 加载任务暂停时的回调
self.room.loadingVM.suspendBlock = ^(BJLLoadingStep step,

BJLLoadingSuspendReason reason,
                                     BJLError *error,
                                     void (^continueCallback)(BOOL isContinue)) {
    bjl_strongify(self);
    // 单步完成，无错误，继续执行下一个加载步骤
    if (reason == BJLLoadingSuspendReason_stepOver) {
        NSLog(@"loading step over: %td", step);
        continueCallback(YES);
        return;
    }

    // 暂停原因
```

```
    NSLog(@"loading step suspend: %td; suspend reason: %td", step,reason);

    // 错误信息
    NSString *message;
    if (reason == BJLLoadingSuspendReason_askForWWANNetwork) {
        message = @"WWAN 网络";
    }
    else if (reason == BJLLoadingSuspendReason_errorOccurred) {
        message = error ? [NSString stringWithFormat:@"%@ - %@",
                           error.localizedDescription,
                           error.localizedFailureReason] : @"错误";
    }

    // 暂停原因为发生错误，不可忽略
    BOOL ignorable = reason != BJLLoadingSuspendReason_errorOccurred;
    // 提示错误信息并提供操作选择
    if (message) {
        UIAlertController *alert = [UIAlertController

alertControllerWithTitle:ignorable ? @"提示" : @"错误"

message:message

preferredStyle:UIAlertControllerStyleAlert];
        [alert addAction:[UIAlertAction
                          actionWithTitle:ignorable ? @"继续" : @"重试"
```

```
                                                style:UIAlertActionStyleDefault
                                                handler:^(UIAlertAction * _Nonnull action) {
                                                    continueCallback(YES);
                                                }]];
        [alert addAction:[UIAlertAction
                                                actionWithTitle:@"取消"
                                                style:UIAlertActionStyleCancel
                                                handler:^(UIAlertAction * _Nonnull action) {
                                                    [self exitRoom];
                                                    continueCallback(NO);
                                                }]];
        [self presentViewController:alert animated:YES completion:nil];
    }
};
```

3.8 大小班切换

教室支持创建分组教室。分组教室可以在仅包含大教室，在大教室内分组互动；也可以支持在小班教室分组互动，回到大班教室一起互动。SDK 支持分组教室，当前用户的分组状态可以通过 `BJLRoom` 的 `loginUser` 的数据获取。对于大小班切换时，SDK 会自动进行切换教室，触发教室各个模块的数据更新，添加的对教室数据的监听将会触发，刷新教室数据和 UI。可以参考 `BJLRoom` 的 `switchingRoom` 状态，给出提示。

```
[self bjl_kvo:BJLMakeProperty(self.room, switchingRoom)
        filter:^BOOL(NSNumber * _Nullable now, id _Nullable old, BJLPropertyChange * _Nullable change) {
```

```
        // bjl_strongify(self);
        return now.boolValue;
    }
    observer:^BOOL(id _Nullable now, id _Nullable old, BJLPropertyChange * _Nullable change) {
        bjl_strongify(self);
        if (self.room.switchingRoom) {
            [self showProgressHUDWithText:@"切换教室中..."];
            [self.overlayViewController hide];
        }
        return YES;
    }];
```

3.9 定制信令

基于客户自身需求，可能需要自己的业务逻辑，SDK 提供发送定制广播信令通道，支持任意用户发送自定义信令，教室内所有用户都可以收取，客户可以实现自己的信令发送，参考 `BJLRoomVM` 。

```
/**
发送定制广播信令
#discussion 发送定制广播信令
#param key    信令类型
#param value   信令内容，合法的 JSON 数据类型 - #see `[NSJSONSerialization isValidJSONObject:]`，序列化成字符串后不能过长，一般不超过 1024 个字符
#param cache   是否缓存，缓存的信令可以通过 `requestCustomizedBroadcastCache:` 方法重新请求
#return BJLError:
```

```
 BJLErrorCode_invalidArguments  不支持的 key，内容为空或者内容过长
 BJLErrorCode_areYouRobot       发送频率过快，要求每秒不超过 5 条、并且每分钟不超过 30 条
 */
BJLError *error = [self.context.room.roomVM sendCustomizedBroadcast:customizedKey value:value cache:isNeedCache];

/**
 收到定制广播信令
 #param key     信令类型
 #param value   信令内容，类型可能是字符串或者字典等 JSON 数据类型
 #param isCache 是否为缓存
 */
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveCustomizedBroadcast:value:isCache:)
        observer:(BJLMethodObserver)^BOOL(NSString *key, id _Nullable _value, BOOL isCache) {
    bjl_strongify(self);
    NSLog(@"didReceiveCustomizedBroadcast %@ %@", key, _value);
}];

/**
 获取定制广播信令缓存
 #discussion 进教室后调用此方法可以获取定制广播信令的缓存，结果回调 `didReceiveCustomizedBroadcast:value:isCache:`
 #param key     信令类型
 #return BJLError:
 BJLErrorCode_invalidArguments  不支持的 key
```

```
31.  BJLErrorCode_areYouRobot      发送频率过快，要求每秒不超过 5 条、并且每分钟不超过 30 条
32.  */
33.  [self.room.roomVM requestCustomizedBroadcastCache:customWebpag
```

3.10 教室信息获取

教室信息可通过 `BJLRoom` 的 `roomInfo` 属性获取，获取时机为进入教室成功（监听到 `enterRoomSuccess` ）之后。

```
1. // 教室信息
2. @property (nonatomic, readonly, copy, nullable) NSObject<BJLRoomInfo> *roomInfo;
```

```
1. // BJLRoomInfo
2. @property (nonatomic, readonly) NSString *ID, *title; // 教室 ID、名称
3. @property (nonatomic, readonly) NSTimeInterval startTimeInterval, endTimeInterval; // 起止时间
4. @property (nonatomic, readonly) BJLRoomType roomType; // 教室类型
```

4. 在线用户信息管理

当前登录用户信息可通过 `BJLRoom` 的 `loginUser` 属性获取，获取时机为进入教室成功（监听到 `enterRoomSuccess` ）之后。

4.1 在线用户列表

在线用户信息列表采用分页加载，参考 BJLOnlineUsersVM。在线的活跃用户信息会在教室进入成功后自动请求，当 `BJLOnlineUsersVM` 的 `activeUsersSynced` 状态变成

YES 之后，活跃的用户数据已经加载完成，可以使用了。通过 `BJLOnlineUsersVM` 的 `onlineTeacher` 和 `currentPresenter` 可以快捷地获取到当前的老师和主讲人。

- 获取当前登录用户信息。

```
// 当前登录用户信息
BJLUser *user = self.room.loginUser;
```

- 监听在线用户变化。

```
[self
bjl_kvo:BJLMakeProperty(self.room.onlineUsersV
 onlineUsers)
  observer:^BOOL(id _Nullable value, id
_Nullable oldValue, BJLPropertyChange *
_Nullable change) {
    bjl_strongify(self);
    [self
updateTitleWithOnlineUsersTotalCount];
    [self.tableView reloadData];
    return YES;
  }];
```

- 加载更多用户。

```
/**
 加载更多在线用户
 #discussion 加载成功更新 `onlineUsers`
 #discussion 参考 `hasMoreOnlineUsersofGroup:`
来获取某一个group是否可以加载更多
 #param count 传 0 默认 20、最多 30
 #return BJLError:
 BJLErrorCode_invalidCalling    错误调用，如
`hasMoreOnlineUsersofGroup:` 为 NO 时调用此方
```

```
法
*/
 if (self.room.onlineUsersVM.hasMoreOnlineUsers
    && [self atTheBottomOfTableView]) {
    [self.room.onlineUsersVM
loadMoreOnlineUsersWithCount:20

groupID:self.room.loginUser.groupID];
 }
```

4.2 用户进入、退出

- 监听用户进入教室。

```
[self
bjl_observe:BJLMakeMethod(self.room.onlineUsersV
 onlineUserDidEnter:)
        observer:^BOOL(BJLUser *user) {
            bjl_strongify(self);
            NSlog(@"onlineUser in: %@",
user.name);
            return YES;
        }];
```

- 监听用户退出教室。

```
[self
bjl_observe:BJLMakeMethod(self.room.onlineUsersV
 onlineUserDidExit:)
        observer:^BOOL(BJLUser *user) {
            bjl_strongify(self);
            NSLog(@"onlineUser out: %@",
user.name);
            return YES;
        }];
```

4.3 分组教室信息

分组信息通过 `BJLOnlineUsersVM` 的属性及方法获取。

- 教室内分组信息。

```
@property (nonatomic, readonly, copy, nullable) NSArray<BJLUserGroup *> *groupList;
```

- 获取各个分组的颜色。

```
/**
 教室内分组颜色
 #param groupID 对应分组的 ID
 #return 十六进制 RGB 色值字符串，如 `#FFFFFF`
 */
 NSString *color = [self.room.onlineUsersVM getGroupColorWithID:group.groupID]
```

- 分组人数变化通知。

```
/**
 学生分组人数变化
 #param groupCountDic <groupID, count> 提供分组及其人数变化
 */
 [self bjl_observe:BJLMakeMethod(self.room.onlineUsersV onlineUserGroupCountDidChange:)
        observer:^BOOL(NSDictionary *groupCountDic) {
    // bjl_strongify(self);
    [groupCountDic enumerateKeysAndObjectsUsingBlock:^(NSString
```

```
*groupID, NSNumber *count, BOOL * _Nonnull stop) {
        NSLog(@"Group %@ has %@ users", groupID, count);
    }];
    return YES;
}];
```

- 分组公告
  - `BJLNotice` 的 `groupNoticeList` 表示教室内各小组的公告，数组元素为 `BJLNoticeModel` 类型；
  - 教室内 `BJLNotice` 公告实例的获取参考文档的公告部分。

4.4 切换主讲人方法

切换主讲人，主讲人只能由教室内最大权限的老师来设置，之后老师本人或者助教才能被设置为主讲人, 参考 `BJLOnlineUsersVM` 。

```
/** 切换主讲
 #discussion 1. 主讲人只能由老师设置，2. 之后老师本人或者助教才能被设置为主讲人
 #param userID  老师或助教的 userID
 #return BJLError:
 BJLErrorCode_invalidCalling    不支持切换主讲，参考 `room.featureConfig.canChangePresenter`
 BJLErrorCode_invalidArguments  错误参数
 BJLErrorCode_invalidUserRole   错误权限，要求老师权限
 */
BJLError *error = [self.room.onlineUsersVM requestChangePresenterWithUserID:userID];
```

4.5 踢出的用户列表

- 踢出用户。

```
/**
 踢出用户
 #param userID 学生ID
 #return BJLError:
 BJLErrorCode_invalidArguments 错误参数；
 BJLErrorCode_invalidCalling 错误调用，如要踢出的用户是老师或助教；
 BJLErrorCode_invalidUserRole 错误权限，要求老师或助教权限。
 */
 BJLError *error = [self.room.onlineUsersVM blockUserWithID:user.ID];
```

- 解除用户被踢出的状态。

```
/**
 解除用户被踢出的状态
 #param userNumber userNumber
 #return BJLError:
 BJLErrorCode_invalidArguments 错误参数；
 BJLErrorCode_invalidCalling 错误调用，如要踢出的用户是老师或助教；
 BJLErrorCode_invalidUserRole 错误权限，要求老师或助教权限。
 */
 BJLError *error = [self.room.onlineUsersVM freeBlockedUserWithNumber:user.number];
```

- 加载被踢出的用户列表

```
    [self bjl_observe:BJLMakeMethod(self.room.onlineUsersV
```

```
didReceiveBlockedUserList:)
        observer:^BOOL(NSArray<BJLUser *>
*userList) {
    bjl_strongify(self);
    self.blockedUserList = [userList
mutableCopy];
    return YES;
}];
```

5. 音视频管理

音视频管理分为 `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**。

5.1 音视频采集

音视频采集功能由 BJLRecordingVM 管理。

5.1.1 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，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。

- 添加采集视图。

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

- 采集音视频开关。

```
/**
 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。

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

5.1.2 采集状态监听

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。

```
@property (nonatomic, readonly) BOOL recordingAudio; // 音频采集开关状态
@property (nonatomic, readonly) BOOL recordingVideo; // 视频采集开关状态
@property (nonatomic, readonly) CGFloat inputVolumeLevel; // 音频输入级别 [0.0 - 1.0]
@property (nonatomic, readonly) CGFloat inputVideoAspectRatio; // 视频采集宽高比
@property (nonatomic, readonly) BOOL forbidRecordingAudio; // 是否禁止当前用户打开音频
@property (nonatomic, readonly) BOOL forbidAllRecordingAudio; // 是否禁止所有人打开音频
```

```
// 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;
    }];
```

- 关键方法监听。

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

recordingDidRemoteChangedRecordingAudio:recorc
        observer:(BJLMethodObserver)^BOOL(BOOL recordingAudio, BOOL recordingVideo, BOOL recordingAudioChanged, BOOL recordingVideoChanged) {
            // bjl_strongify(self);
            NSString *message = @"";
            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;
        }];
```

5.1.3 外接移动端设备作为摄像头采集

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;
```

5.1.4 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;
```

```
/**
 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;
```

- 清晰度设置。

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

```
3.
4. /**
5.  改变视频清晰度
6.  #param videoDefinition 清晰度
7.  #return BJLError:
8.  BJLErrorCode_invalidCalling  错误调用
9.  */
10. BJLError *error = [self.room.recordingVM
    updateVideoDefinition:BJLVideoDefinition_high];
```

- 美颜设置。

```
1. /** 美颜，默认关闭 */
2. @property (nonatomic, readonly)
   BJLVideoBeautifyLevel videoBeautifyLevel;
3.
4. /**
5.  改变美颜等级
6.  #param videoBeautifyLevel videoBeautifyLevel
7.  #return BJLError:
8.  BJLErrorCode_invalidCalling  错误调用
9.  */
10. [self.room.recordingVM
    updateVideoBeautifyLevel:BJLVideoBeautifyLevel_or
```

5.1.5 获取 PCM 格式的音频数据

```
1. /** 是否开启 PCM 音频数据回调，默认为 NO，需要在
   进入教室前设置，不支持进入教室后修改 */
2. @property (nonatomic) BOOL enablePCMData;
3. /**
4.  采集到的音频数据，仅支持 AVSDK 的班型
5.  #param length 音频数据长度
6.  #param data PCM 音频数据
7.  */
```

```
8. -(BJLObservable)recordingAudioPCMDataDidUpdate:(uint8_t [_Nullable])data length:(int)length;
```

5.2 音视频播放

音视频播放功能由 BJLPlayingVM 管理。

5.2.1 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。

2.0 及以上版本的 BJLiveCore SDK 支持同时播放一个用户的多路视频流，由 `BJLPlayingVM` 管理，对比旧版本 SDK，API 改动较大，集成方如果是从 1.x 版本或 `2.0.0-alpha`、`2.0.0-beta` 升级上来，且没有多视频流的需求，可

以使用 BJLPlayingAdapterVM 管理音视频播放，此适配层支持旧版的单路流及主辅摄像头双路流模板，API 相对改动较小，具体使用方式可参考 旧版本 SDK 音视频模板适配 部分。

多路流模板中用户音视频流对应关系如下：

用户音视频流实例	视频源类型	
BJLMediaUser	BJLMediaSource_mainCamera	主
	BJLMediaSource_screenShare	屏
	BJLMediaSource_mediaFile	媒
	BJLMediaSource_camera	辅
	BJLMediaSource_screenShare	辅

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。

5.2.2 监听播放信息

音视频信息通过监听 `self.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
 #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 SDK 会处理音视频打断、恢复、前后台切换等情况
 */
```

```
@property (nonatomic, readonly, copy, nullable)
NSArray<BJLMediaUser *> *playingUsers;
```

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

- 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。

```
/**
 扩展音视频流用户列表
 #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，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;
    }];
```

- 监听老师播放媒体文件、共享桌面。

通过监听 `BJLPlayingVM` 的 `teacherSharingDesktop`、`teacherPlayingMedia` 变化实现。

在 2.x 版本 SDK 中，可以通过监听 `playingUserDidUpdate:old:` 方法调用来代替，与监听这两个属性相比，能通过方法回调的 `BJLMediaUser` 实例直接获取到老师最新的音视频信息，使用 `BJLMediaSource` 判断媒体类型，参考监听用户开关音视频 的示例代码。

```
// example：监听老师开关媒体文件播放
```

```
[self
bjl_kvo:BJLMakeProperty(self.room.playingVM,
teacherPlayingMedia)
      filter:^BOOL(NSNumber * _Nullable value,
NSNumber * _Nullable oldValue,
BJLPropertyChange * _Nullable change) {
         return value.boolValue !=
oldValue.boolValue;
      }
    observer:^BOOL(NSNumber * _Nullable value,
NSNumber * _Nullable oldValue,
BJLPropertyChange * _Nullable change) {
        // bjl_strongify(self);
        NSLog(@"老师%@了媒体文件播放",
value.boolValue ? @"开启" : @"关闭");
        return YES;
    }];

// example：监听老师开关屏幕共享
[self
bjl_kvo:BJLMakeProperty(self.room.playingVM,
teacherSharingDesktop)
      filter:^BOOL(NSNumber * _Nullable value,
NSNumber * _Nullable oldValue,
BJLPropertyChange * _Nullable change) {
         return value.boolValue !=
oldValue.boolValue;
      }
    observer:^BOOL(NSNumber * _Nullable value,
NSNumber * _Nullable oldValue,
BJLPropertyChange * _Nullable change) {
        // bjl_strongify(self);
        NSLog(@"老师%@了屏幕共享",
value.boolValue ? @"开启" : @"关闭");
        return YES;
    }];
```

5.2.3 播放控制

- 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
 */

// 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，例如
 | self.room.playingVM.autoPlayVideoBlock = ^BJLAutoPlayVideo(BJLMediaUser *user, NSInteger cachedDefinitionIndex) {
 |     BOOL autoPlay = user.number && ![self.autoPlayVideoBlacklist containsObject:user.number];
 |     NSInteger definitionIndex = cachedDefinitionIndex;
```

```
|     if (autoPlay) {
|         NSInteger maxDefinitionIndex = MAX(0, (NSInteger)user.definitions.count - 1);
|         definitionIndex = (cachedDefinitionIndex <= maxDefinitionIndex
|                            ? cachedDefinitionIndex : maxDefinitionIndex);
|     }
|     return BJLAutoPlayVideoMake(autoPlay, definitionIndex);
| };
*/
@property (nonatomic, copy, nullable) BJLAutoPlayVideo (^autoPlayVideoBlock)(BJLMediaUser *user, NSInteger cachedDefinitionIndex);

// 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
[self.room.playingVM updatePlayingUserWithID:playingUser.ID videoOn:NO mediaSource:playingUser.mediaSource];

// 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
// 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
[self.autoPlayVideoBlacklist addObject:[self videoKeyForUser:playingUser] ?: @""];
```

```
// example：自动播放不在黑名单中的视频流
self.room.playingVM.autoPlayVideoBlock = ^BJLAutoPlayVideo(BOOL autoPlay, NSInteger
```

```
definitionIndex)(BJLMediaUser *user, NSInteger
cachedDefinitionIndex) {
    bjl_strongify(self);
    NSString *videoKey = [self
videoKeyForUser:user];
    // 不在黑名单中的视频流，自动播放
    BOOL autoPlay = videoKey && !
[self.autoPlayVideoBlacklist
containsObject:videoKey];
    // 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);
};
```

- 获取正在播放的视频流的清晰度。

```
/**
 获取播放用户的清晰度
 #param userID 用户 ID
 #param mediaSource 视频源类型
 #return 播放时传入的 `definitionIndex`
 */

// example：获取老师主摄像头视频的清晰度序号，该
序号为清晰度在 `BJLMediaUser` 的 `definitions` 数
```

```
组中的序号
NSInteger definitionIndex = [self.room.playingVM
definitionIndexForUserWithID:self.room.onlineUsers

mediaSource:BJLMediaSource_mainCamera];
```

- 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];
```

5.2.4 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。

列表覆盖更新一般在 进入教室 或 断网重连 时回调。

```
/**
 `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
```

```
16.                         && now.mediaSource ==
BJLMediaSource_mainCamera);
17.             BOOL videoChanged = (now.videoOn
!= old.videoOn);
18.
19.             NSString *videoTitle =
BJLVideoTitleWithMediaSource(now.mediaSource);
20.             if (audioChanged && videoChanged) {
21.                 if (now.audioOn && now.videoOn) {
22.                     [self showProgressHUDWithText:
[NSString stringWithFormat:@"老师开启了麦克风
和%@", videoTitle]];
23.                 }
24.                 else if (now.audioOn) {
25.                     [self
showProgressHUDWithText:@"老师开启了麦克风"];
26.                 }
27.                 else if (now.videoOn) {
28.                     [self showProgressHUDWithText:
[NSString stringWithFormat:@"老师开启了%@",
videoTitle]];
29.                 }
30.                 else {
31.                     [self showProgressHUDWithText:
[NSString stringWithFormat:@"老师关闭了麦克风
和%@", videoTitle]];
32.                 }
33.             }
34.             else if (audioChanged) {
35.                 if (now.audioOn) {
36.                     [self
showProgressHUDWithText:@"老师开启了麦克风"];
37.                 }
38.                 else {
39.                     [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，建议直接使用 `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
 #discussion 用户视频播放成功
 #param playingUser 播放的视频对应的用户
 */
[self
bjl_observe:BJLMakeMethod(self.room.playingVM,
playingUserDidFinishLoadingVideo:)
        observer:^BOOL(BJLUser *user) {
            bjl_strongify(self);
            [self
tryToCloseLoadingViewWithUser:user];
            return YES;
        }];
```

- 播放出现卡顿。

```
/**
 播放出现卡顿
 #param user 出现卡顿的正在播放的视频用户实例
 */
[self
bjl_observe:BJLMakeMethod(self.context.mediaPlay
 playLagWithPlayingUser:)
```

```
        observer:(BJLMethodObserver)^BOOL(BJLMediaUser *user){
            // bjl_strongify(self);
            NSLog(@"%@的音视频出现卡顿", user.name);
            return YES;
        }];
```

5.3 音视频设置

音视频流由 BJLMediaVM 管理。

5.3.1 音视频链路设置

音视频链路分为 上行链路 和 下行链路 两种，上行链路表示发言用户将自己的音视频数据流推送到服务端所采用的链路，而下行链路则是拉取发言用户的音视频数据流、进行播放时所采用的链路。

上、下行链路按类型划分为 UDP 和 TCP 两种。基于 UDP 的 RTP/RTCP 等协议，延迟小于300ms，延迟可控，一般都是自己搭建服务器来实现；基于 TCP 的 RTMP，延迟比较大（3-5 秒），延迟不可控，一般都是用 CDN 来实现。**不作配置的情况下，默认使用 UDP**。

音视频链路设置仅 **非 webRTC 教室** 可用。

在 `BJLMediaVM` 类中定义了 `upLinkType` 表示上行链路类型，`downLinkType` 表示下行链路类型，均为 `BJLLinkType` 枚举类型。

```
/**
 是否允许设置上、下行链路类型
 #discussion `upLinkTypeReadOnly`/`downLinkTypeReadOnly` 为 YES 时设置 `upLinkType`/`downLinkType` 无效
```

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

// 设置用于采集音视频的上行链路为 UDP
[self.room.mediaVM updateUpLinkType:BJLLinkType_UDP];
// 设置用于播放音视频的下行链路为 TCP
[self.room.mediaVM updateDownLinkType:BJLLinkType_TCP];
```

- TCP 上下行链路控制。

```
// TCP 上行 CDN 切换，下行类似，参考 `BJLMediaVM`

/** TCP 上行线路可用 CDN 数 */
@property (nonatomic, readonly) NSUInteger availableUpLinkCDNCount;

/** TCP 上行线路优先使用的 CDN index
 #discussion 调用用 updatePrefferedCDNWithIndex: 设置该值
 #discussion 指定线路推流时可设置该值为范围 [0, availableCDN] 内的整数，每一个数对应一个 CDN 线路。指定 CDN 不可用时，服务器将自动分配。
 #discussion 设置该值为 [0, availableCDN] 外的任意值代表不指定 CDN，由服务器自动分配。
 #discussion 默认值为 NSNotFound, 即服务器自动分配。
 #discussion 改变该值将导致重新推流
 */
@property (nonatomic, readonly) NSInteger upLinkCDNIndex;

```

```
/** 设置 upLinkCDNIndex 并重新推流
 #discussion 调用该方法会将上行链路切换到 TCP
 */
BJLError *error = [self.room.mediaVM updateTCPUpLinkCDNWithIndex:selectIndex];
```

5.3.2 音视频采集，播放控制等配置

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配置控制。

```
/** 当前应用是否控制音频 */
@property (nonatomic, readonly) BOOL isAudioSessionActive;

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

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

/** 是否播放视频静音, 默认不静音 */
@property (nonatomic, readonly) BOOL needMutePlayingAudio;
- (BJLError *)updateNeedMutePlayingAudio:(BOOL)needMutePlayingAudio;
```

5.3.3 webRTC 教室回调

对于 `webRTC` 教室（通过 `self.room.featureConfig.isWebRTC` 判断），`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;
        }];

// 音视频网络状态更新回调
[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){
43.             bjl_strongify(self);
44.             // 处理音量变化
```

```
            [self updateVolume:volume forUser:user];
            return YES;
        }];
```

5.4 专业小班课 API

- 视频窗口位置更新。

```
/**
 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。

```
// 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);
            NSLog(@"上台请求被拒绝：%@",
(responseCode == 2)? @"该学生已离开教室" : @"上
台人数已满");
            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;
        }];
```

6. 旧版 SDK 音视频模板适配 API

2.0 及以上版本的 BJLiveCore SDK 支持同时播放一个用户的多路视频流，由 `BJLPlayingVM` 管理，对比旧版本 SDK，API 改动较大，集成方如果是从 1.x 版本升级上来，且没有多视频流的需求，可以使用 BJLPlayingAdapterVM 管理音视频播放，此适配层在 2.x SDK `BJLPlayingVM` 的多路流模板 API 的基础上进行适配封装，支持旧版 SDK 的单路流及主辅摄像头双路流模板，API 相对改动较小。

6.1 音视频模板适配要点说明

- 单路流模板。

1.x 版本 SDK 中，每个发言用户最多存在一路音视频流，2.x 版本针对这种使用场景进行了适配，通过 `BJLPlayingAdapterVM` 适配层提供近似于旧版 SDK 的 API，实现相关功能。单路流模板中，可通过 `self.room.mainPlayingAdapterVM` 管理音视频播放, 与 1.x 版本的 `self.room.playingVM` 功能相同。

单路流模板中用户音视频流对应关系如下：

用户音视频流实例	视频源类型	
BJLMediaUser	BJLMediaSource_camera	主
	BJLMediaSource_screenShare	屏
	BJLMediaSource_mediaFile	媒

对于一个 `BJLUser` 实例，在 `self.room.mainPlayingAdapterVM.playingUsers` 数组中最多存在一个 `BJLMediaUser` 实例， `mediaSource` 为 `BJLMediaSource_mainCamera` 、 `BJLMediaSource_screenShare` 或 `BJLMediaSource_mediaFile` 。

1.x 版本 SDK 升级到 2.x 时，代码改动示例如下：

```
// 1.x 版本播放对象用户视频
[self.playingVM updatePlayingUserWithID:user.ID videoOn:YES];

// 升级到 2.x 版本之后，使用 mainPlayingAdapterVM 代替 playingVM
[self.mainPlayingAdapterVM updatePlayingUserWithID:user.ID videoOn:YES];
```

- 主辅摄像头双路流模板。

`2.0.0-alpha` 及 `2.0.0-beta` 版本的 SDK 支持双路流模板，每个发言用户最多可同时存在主摄像头、辅主摄像头两路

流，后续的 2.x 版本也同样通过 `BJLPlayingAdapterVM` 针对这种使用场景进行了适配。使用 `self.room.mainPlayingAdapterVM` 管理主摄像头音视频，`self.room.extraPlayingAdapterVM` 管理辅助摄像头音视频，两者的 `BJLMediaUser` 实例通过 `cameraType` 属性区分, 参考 `BJLCameraType` 。

双路流模板中用户音视频流对应关系如下：

用户音视频流实例	摄像头位置	
BJLMediaUser	BJLCameraType_main	BJLMediaS
		BJLMediaS
		BJLMediaS
	BJLCameraType_extra	BJLMediaS
		BJLMediaS

双路流模板中，对于每个发言用户，他的主摄像头流对应的 `BJLMediaUser` 实例存在于 `self.room.mainPlayingAdapterVM.playingUsers` 中，实例的 `cameraType` 为 `BJLCameraType_main`，`mediaSource` 为 `BJLMediaSource_mainCamera`、`BJLMediaSource_screenShare` 或 `BJLMediaSource_mediaFile`，每个发言用户在数组中有且仅存在一个实例；辅助摄像头流对应的 `BJLMediaUser` 实例存在于 `self.room.mainPlayingAdapterVM.playingUsers` 中，实例的 `cameraType` 为 `BJLCameraType_extra`，`mediaSource` 为 `BJLMediaSource_extraCamera` 或 `BJLMediaSource_extraScreenShare`，每个发言用户在数组中最多存在一个实例。

6.2 监听播放信息

适配 API 中，主摄像头的音视频信息通过监听 `self.room.mainPlayingAdapterVM` 的属性变化获取，辅助摄像头的音视频信息通过监听 `self.room.extraPlayingAdapterVM` 的属性变化获取。

- 监听音视频用户列表。

`self.room.mainPlayingAdapterVM` 的 `playingUsers` 表示主摄像头音视频用户列表，`self.room.extraPlayingAdapterVM` 的 `playingUsers` 表示辅助摄像头音视频用户列表。

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

// example：监听 mainPlayingAdapterVM 的 playingUsers 变化，获取主摄像头音视频的实时信息。
[self bjl_kvo:BJLMakeProperty(self.room.mainPlayingAda
 playingUsers)
    observer:^BOOL(id _Nullable value, id _Nullable oldValue, BJLPropertyChange * _Nullable change) {
        // bjl_strongify(self);
        NSLog(@"playing users changed");
        return YES;
```

```
    }];
```

- 监听当前正在播放的视频对应的用户列表。

`self.room.mainPlayingAdapterVM` 的 `videoPlayingUsers` 表示当前正在播放的主摄像头视频用户对应的列表，`self.room.extraPlayingAdapterVM` 的 `videoPlayingUsers` 表示当前正在播放的辅助摄像头视频对应的用户列表。

```
/**
 正在播放的视频用户
 #discussion `playingUsers` 的子集
 #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 mainPlayingAdapterVM 的 videoPlayingUsers 变化
[self bjl_kvo:BJLMakeProperty(self.room.mainPlayingAdapterVM, videoPlayingUsers)
    observer:^BOOL(NSArray<BJLUser *> *value, NSArray<BJLUser *> *oldValue, BJLPropertyChange * _Nullable change) {
        NSLog(@"当前正在播放%td人的主摄像头视频", value.count);
        return YES;
    }];
```

6.3 播放控制

主摄像头的播放通过 `self.room.mainPlayingAdapterVM` 控制；辅助摄像头的播放通过

`self.room.extraPlayingAdapterVM` 控制。

- 获取播放视图。

```
/**
 播放出现卡顿
 #param user 出现卡顿的正在播放的视频用户实例
 */
- (nullable UIView *)playingViewForUserWithID:(NSString *)userID;
```

```
// example：获取老师主摄像头位置（BJLCameraType_main）视频的视图
UIView *mainCameraView = [self.room.mainPlayingAdapterVM playingViewForUserWithID:self.room.onlineUsersVM

// example: 获取老师辅助摄像头位置（BJLCameraType_extra）视频的视图
UIView *extraCameraView = [self.room.extraPlayingAdapterVM playingViewForUserWithID:self.room.onlineUsersVM
```

- 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()`
 #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;
- (nullable BJLError *)updatePlayingUserWithID:(NSString *)userID
                                       videoOn:(BOOL)videoOn
                                 definitionIndex:(NSInteger)definitionIndex;
```

```
// example：关闭 user 对应的主摄像头视频流，user 为 `BJLMediaUser` 实例
[self.room.mainPlayingAdapterVM updatePlayingUserWithID:user.ID
                                                videoOn:NO];
```

- 自动播放视频并指定清晰度回调。

主摄像头通过 `self.room.mainPlayingAdapterVM.autoPlayVideoBlock` 回调，辅助摄像头通过 `self.room.mainPlayingAdapterVM.extraPlayVideoBlock` 。示例可参考 BJLPlayingVM 的对应示例代码。

```
/**
 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，例如
 | self.room.playingVM.autoPlayVideoBlock =
^BJLAutoPlayVideo(BJLMediaUser *user,
NSInteger cachedDefinitionIndex) {
 |     BOOL autoPlay = user.number && !
[self.autoPlayVideoBlacklist
containsObject:user.number];
 |     NSInteger definitionIndex =
cachedDefinitionIndex;
 |     if (autoPlay) {
 |         NSInteger maxDefinitionIndex = MAX(0,
(NSInteger)user.definitions.count - 1);
 |         definitionIndex = (cachedDefinitionIndex
<= maxDefinitionIndex
 |                            ? cachedDefinitionIndex :
maxDefinitionIndex);
 |     }
 |     return BJLAutoPlayVideoMake(autoPlay,
definitionIndex);
 | };
 */
@property (nonatomic, copy, nullable)
BJLAutoPlayVideo (^autoPlayVideoBlock)
(BJLMediaUser *user, NSInteger
cachedDefinitionIndex);
```

- 获取正在播放的视频流的清晰度。

```
/** 获取播放用户的清晰度
 #param userID 用户 ID
 #return 播放时传入的 `definitionIndex`
 */
- (NSInteger)definitionIndexForUserWithID:
(NSString *)userID;
```

6.4 监听播放回调

主摄像头的播放回调信息通过监听 `self.room.mainPlayingAdapterVM` 的方法调用获取；辅助摄像头的播放回调信息通过监听 `self.room.extraPlayingAdapterVM` 的方法调用获取。如果支持双路流，则需要分别监听。监听代码示例可参考 BJLPlayingVM 的对应部分，只需要将监听的对象改为 `mainPlayingAdapterVM` 或 `extraPlayingAdapterVM` 即可;

- 音视频用户列表覆盖更新。

列表覆盖更新一般在 进入教室 或 断网重连 时回调。

```
/**
 `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
 */
- (BJLObservable)playingUsersDidOverwrite:(nullable NSArray<BJLMediaUser *> *)playingUsers;
```

- 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
 */
- (BJLObservable)playingUserDidUpdate:(nullable BJLMediaUser *)now
                                  old:(nullable BJLMediaUser *)old;
```

- 用户视频清晰度变化。

```
/**
 用户开改变视频清晰度
 #param now 新用户信息
 #param old 旧用户信息
 */
- (BJLObservable)playingUserDidUpdateVideoDefiniti(nullable BJLMediaUser *)now
                                                old:(nullable BJLMediaUser *)old;
```

- 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 用户实例
 */
- (BJLObservable)playingViewAspectRatioChanged:
```

```
(CGFloat)videoAspectRatio
                                forUser:(BJLMediaUser
*)user;
```

- 用户视频流加载状态。

可用于显示视频 loading 状态。

```
/**
 将要播放视频
 #discussion 播放或者关闭视频的方法被成功调用
 #param playingUser 将要播放视频用户
 */
-
(BJLObservable)playingUserDidStartLoadingVideo:
(nullable BJLMediaUser *)playingUser;

/**
 播放成功
 #discussion 用户视频开启或者关闭成功
 #param playingUser 播放视频的用户
 */
-
(BJLObservable)playingUserDidFinishLoadingVideo:
(nullable BJLMediaUser *)playingUser;
```

- 播放出现卡顿。

```
/**
 播放出现卡顿
 #param user 出现卡顿的正在播放的视频用户实例
 */
- (BJLObservable)playLagWithPlayingUser:
(BJLMediaUser *)user;
```

7. 举手发言

7.1 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。

- 判断老师是否在教室。

```
BOOL hasTeacher = !!self.room.onlineUsersVM.onlineTeacher;
```

- 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，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
 */
bjl_weakify(self);
[self
bjl_observe:BJLMakeMethod(self.room.speakingReq

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
*/
[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.speakingRequest, 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];
```

7.2 学生处理发言邀请

学生还可以收到老师的发言邀请（移动端目前不支持发送发言邀请），接受之后将进入发言状态。

- 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.speakingReq 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
```

7.3 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。

```
1. [self
   bjl_kvo:BJLMakeProperty(self.room.speakingReques
    speakingRequestUsers)
2.     options:NSKeyValueObservingOptionNew |
   NSKeyValueObservingOptionOld
3.     observer:^BOOL(NSArray<BJLUser *> *
   _Nullable value, NSArray<BJLUser *> * _Nullable
   oldValue, BJLPropertyChange * _Nullable change)
   {
4.         NSLog(@"value: %lu elements, oldValue:
   %lu elements", (unsigned long)value.count,
   (unsigned long)oldValue.count);
5.         return YES;
6.     }];
```

- 允许 / 拒绝发言。

```
1. [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。

```
1. bjl_weakify(self);
```

```
[self
bjl_observe:BJLMakeMethod(self.room.speakingReq
 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];
```

8. 课件

课件包括白板、PPT和画笔，SDK 支持 pdf、word、动效 PPT 等文档的显示，教室里面默认至少有一个白板课件，上传课件支持图片，PDF文件，Word文件，PPT文件。2.x的 SDK 支持多文档实例，`BJLSlideVM`，`BJLSlideshowVM` 移除，替换为 `BJLDocumentVM`。

- 设置课件类型**(需要在进入教室之前设置)**：SDK 提供 native 和 H5 两种类型的课件。native 课件加载快、支持缩放手势，但不支持 PPT 动画；H5 课件加载略慢，不支持缩放手势，支持 PPT 动画。**默认使用 H5 课件**。目前 SDK 支持课件的动态切换，在使用 H5 课件的情况下，教室里存在动态课件，将会使用 H5 课件，教室里只有静态课件的时候，将会使用 native 课件。`BJLiveCore` **2.1.0** 版本支持动态开关 PPT 动效，设置 `BJLRoom` 的 `disablePPTAnimation` 可即时切换课件类型。

```
// 设置课件类型，不设置则默认使用 H5 课件
self.room.disablePPTAnimation = NO; // YES：native, NO：H5
```

8.1 单实例课件控制器显示

单实例课件控制器，所有课件都用一个控制器管理。支持设置的参数参考 `BJLSlideshowUI`。

```
/**
 静态课件尺寸
 加载课件图片时对图片做等比缩放，长边小于/等于 `imageSize`，放大时加载 1.5 倍尺寸的图片
 单位为像素，默认初始加载 720、放大加载 1080，取值在 `BJLAliIMGMinSize` 到 `BJLAliIMGMaxSize` 之间 (1 ~ 4096)
```

```
 不建议进教室成功后设置此参数，因为会导致已经加载过的图片缓存失效
 只对静态课件生效，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
self.room.slideshowViewController.imageSize = 720;
```

```
// 显示课件视图, 将 BJLRoom 的课件视图添加到当前 viewController 的对应视图
[self addChildViewController:self.room.slideshowViewCo
[self.slideshowView addSubview:self.room.slideshowViewController.viev
[self.room.slideshowViewController didMoveToParentViewController:self];
```

```
/**
 静态课件显示模式
 只支持 BJLContentMode_scaleAspectFit, BJLContentMode_scaleAspectFill
 只对静态课件生效，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
self.room.slideshowViewController.contentMode = BJLContentMode_scaleAspectFit;
```

8.2 多实例课件控制器显示

每一个课件都可以创建一个单独的控制器管理。支持设置的参数参考 `BJLSlideshowUI` 。

- 显示控制。

```
/** 黑板视图控制器 */
@property (nonatomic, readonly) UIViewController<BJLBlackboardUI> *blackboardViewController;
/** 黑板页数 */
@property (nonatomic, readonly) NSInteger blackboardContentPages;
/** 设置黑板视图（blackboardViewController）的默认背景图 */
@property (nonatomic) UIImage *blackboardImage;
// 获取当前黑板页码
CGFloat localPageIndex = self.room.documentVM.blackboardViewController.pa

/**
 指定文档 ID 创建对应的视图控制器
 #param documentID 文档 ID，通过 BJLDocument 的 documentID 获得
 #return 对应文档的视图控制器
 */
 self.documentViewController = [room.documentVM documentViewControllerWithID:documentID];
```

- 同步以及更新文档显示状态。

```

/**
 更新文档显示信息
 #param documentID 文档 ID
 #param displayInfo 文档显示信息
 */
[self.context.room.documentVM updateDocumentWithID:BJLBlackboardID
```

```
displayInfo:displayInfo];

/**
 更新文档显示信息的通知
 #param document 更新后的文档信息
 */
[self
bjl_observe:BJLMakeMethod(self.context.room.docu
 didUpdateDocument:)
        filter:^BOOL(BJLDocument *document){
            // bjl_strongify(self);
            return [document.documentID
isEqualToString:BJLBlackboardID];
        }
       observer:^BOOL(BJLDocument *document)
{
            bjl_strongify(self);
            // 黑板滑动
            [self
scrollToTopOffset:document.displayInfo.topOffset];
            return YES;
        }];

/**
 更新文档窗口
 #param documentID 文档 ID
 #param action 更新类型，参考
BJLWindowsUpdateModel 的
BJLWindowsUpdateAction
 #param displayInfos 教室内所有文档窗口显示信息
 #return 调用错误, 参考 BJLErrorCode
 */
[self.room.documentVM
updateDocumentWindowWithID:documentID
                                action:action
```

```
displayInfos:self.documentWindowDisplayInfos];

/**
 文档窗口更新通知
 #param updateModel 更新信息
 #param shouldReset 是否重置
 */
[self
bjl_observe:BJLMakeMethod(self.room.documentVM

didUpdateDocumentWindowWithModel:shouldReset
        observer:
(BJLMethodObserver)^BOOL(BJLWindowUpdateMod
 *updateModel, BOOL shouldReset) {
            bjl_strongify(self);
            if (shouldReset) {
                [self
resetDocumentWindowsWithModel:updateModel];
            }
            else {
                [self
updateDocumentWindowWithModel:updateModel];
            }
            return YES;
        }];
```

8.3 课件控制器管理

课件控制器实现了 `BJLSlideshowUI` 协议，提供了通用的 API，加载图片尺寸，翻页，备注，画笔状态，能否翻页状态等。也有仅静态课件可用和仅动态课件可用 API。

- 共通 API。

```objc
/** 当前课件视图类型，不固定，可能发生改变 */
@property (nonatomic, readonly) BJLPPTViewType viewType;
/** 画笔开关状态 */
@property (nonatomic, readonly) BOOL drawingEnabled;
/** 当前课件是否支持滑动翻页 */
@property (nonatomic, readonly) BOOL scrollEnabled;
/** 是否显示 PPT 备注 */
@property (nonatomic, readonly) BOOL showPPTRemarkInfo;
/** 能否向前翻页，或者存在前一步 */
@property (nonatomic, readonly) BOOL canStepForward;
/** 能否向后翻页，或者存在后一步 */
@property (nonatomic, readonly) BOOL canStepBackward;
/**
 本地当前页、可能与教室内的页数不同
 参考 `BJLDocumentVM` 的 `currentSlidePage.documentPageIndex`
 不能滑动翻页时，参考 `scrollEnabled`，设置 pageIndex 无效 */
@property (nonatomic, readonly) NSInteger pageIndex;
- (nullable BJLError *)updatePageIndex:(NSInteger)pageIndex;
/** 页面控制按钮 */
@property (nonatomic) UIButton *pageControlButton;
/** 课件当前页图像在 PPT 视图中的布局信息 */
@property (nonatomic) CGRect imageFrameInPPTView;
/**
```

```
 适用于单文档: 是否禁用跨越文档翻页，默认允许跨文档，
 不同文档之间的 pageStepForward 和 pageStepBackward 是可用的
 如果禁用，在临界处将 pageStepForward 和 pageStepBackward 不可用，只能通过设置页码 index 进行翻页
 */
@property (nonatomic) BOOL disableCrossDoc;
/** 动态PPT加载失败时,是否要切静态PPT */
@property (nonatomic, copy, nullable) void (^shouldSwitchNativePPTBlock)(NSString *_Nullable documentID, void (^callback)(BOOL shouldSwitch));
/** 向前翻页 */
- (nullable BJLError *)pageStepForward;
/** 向后翻页 */
- (nullable BJLError *)pageStepBackward;
/** 清除白板 */
- (void)clearDrawing;
/**
 设置是否显示 PPT 备注
 #param show 是否显示
 */
- (void)updateShowPPTRemarkInfo:(BOOL)show;
/**
 是否可以滑动翻页，默认可滑动翻页，参考 `scrollEnabled` 属性
 #param scrollEnabled 是否可滑动翻页
 #discussion 对于动态课件，优先级低于画笔状态和动态课件交互状态
 #discussion 开启画笔或者动态课件可交互的情况下，仍然不能滑动翻页
 #discussion 助教 `BJLRoomVM` 获取 `getAssistantaAuthorityWithDocumentControl` 无权限时，无法滑动翻页
```

```
 #discussion 学生 `BJLDocumentVM.forbidStudentChangePPT` 被禁用时，无法滑动翻页
 #discussion 学生 `BJLFeatureConfig.disableStudentChangePPTPage` 被禁用时，无法滑动翻页
 #discussion 小班课学生 `BJLDocumentVM.authorizedPPT` 未授权时，无法滑动翻页
 #discussion 当前课件是否可以滑动会受老师端影响，老师端可以控制学生和助教的课件权限，
 #discussion 如果需要在代码中保持禁止滑动，可以监听 scrollEnabled 的值进行设置，或者直接禁用视图交互
 */
- (void)updateScrollEnabled:(BOOL)scrollEnabled;
```

- 静态课件 API。

```
/**
 静态课件显示模式
 只支持 BJLContentMode_scaleAspectFit, BJLContentMode_scaleAspectFill
 只对静态课件生效，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
@property (nonatomic) BJLContentMode contentMode;
/**
 静态课件尺寸
 加载课件图片时对图片做等比缩放，长边小于/等于 `imageSize`，放大时加载 1.5 倍尺寸的图片
 单位为像素，默认初始加载 720、放大加载 1080，取值在 `BJLAliIMGMinSize` 到 `BJLAliIMGMaxSize` 之间 (1 ~ 4096)
```

```
 不建议进教室成功后设置此参数，因为会导致已经加载过的图片缓存失效
 只对静态课件生效，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
@property (nonatomic) NSInteger imageSize;
/**
 静态课件占位图
 只对静态课件生效，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
@property (nonatomic) UIImage *placeholderImage;
/**
 静态课件是否支持缩放
 #param scaleEnabled 是否支持缩放
 #discussion 默认支持缩放
 */
- (void)updateScaleEnabled:(BOOL)scaleEnabled;
```

- 动态课件 API。

```
/** 动态课件加载成功 */
@property (nonatomic, readonly) BOOL webPPTLoadSuccess;
/** 动态课件交互状态，目前仅支持小班课 */
@property (nonatomic, readonly) BOOL webPPTInteractable;
/** 动态课件翻页指示图标
 只对动画课件，参考 `BJLRoom` 的 `disablePPTAnimation`
 */
@property (nonatomic) UIImage *prevPageIndicatorImage, *nextPageIndicatorImage;
```

```
/**
 动态课件手势触发的翻页是否能够向前翻页，默认能翻页
 不能滑动翻页时，参考 `scrollEnabled`，也不能手势翻页 */
@property (nonatomic) BOOL pageGestureCanStepForward;
/**
 动态课件手势触发的翻页是否能够向后翻页，默认能翻页
 不能滑动翻页时，参考 `scrollEnabled`，也不能手势翻页 */
@property (nonatomic) BOOL pageGestureCanStepBackward;
/**
 动态课件是否可以交互
 #param interactable 是否可交互
 #discussion 不能滑动翻页时，参考 `scrollEnabled`，也无法交互
 */
- (void)updateWebPPTInteractable:(BOOL)interactable;
```

8.4 上传、添加课件，课件管理

- 上传图片格式课件。

```
bjl_weakify(self);
[self.room.documentVM uploadImageFile:fileURL
                             progress:^(CGFloat progress){
                                 bjl_strongify(self);
                                 // 显示进度
                                 self.progressView.progress = progress;
                             }
```

```
                finish:^(BJLDocument *
_Nullable document, BJLError * _Nullable error) {
                    bjl_strongify(self)
                    if(document){
                        [self.room.documentVM
addDocument:document];
                    }
                    else{
                        NSLog(@"error:%@",
error);
                    }
                }];
```

- 上传更多文件格式课件。

```
// 上传
[self.room.documentVM uploadFile:url
                        mimeType:mimeType
                        fileName:name
                     isAnimated:isAnimated
                        progress:^(CGFloat progress) {
                            //bjl_strongify(self);
                            NSLog(@"progress", progress);
                        }
                         finish:^(BJLDocument * _Nullable
document, BJLError * _Nullable error) {
                            bjl_strongify(self);
                            if (!error) {
                                if (image) {
                                    // 图片不需要转码，直接添加
到教室
                                    [self.room.documentVM
addDocument:document];
                                }
                                else {
```

```
                                // 开始轮询转码进度
                                [self startPollTimer];
                            }
                        }
                        else {
                            errorMessage = @"文件上传失败\n无法使用";
                        }
                    }];

// 转码
[self.room.documentVM requestTranscodingProgressWithFileIDList:array completion:^(NSArray<BJLDocumentTranscodeModel *> * _Nullable transcodeModelArray, BJLError * _Nullable error) {
bjl_strongify(self);
                                    if (error) {
                                        return;
                                    }
                                    for (BJLDocumentTranscodeModel *model in transcodeModelArray) {
                                        if (model.progress >= 100) {
                                            // 转码完成，请求转码完成之后的新文档
                                            [self requestDocumentList];
                                        }
                                        else {
                                            // 更新转码进度
```

```
41.
NSLog(@"progress", progress);
42.                                         }
43.                                     }
44.                                 }];
45. // 添加文档
46. [self.room.documentVM
requestDocumentListWithFileIDList:@[fileID]
47.
completion:^(NSArray<BJLDocument *> *
_Nullable documentArray, BJLError * _Nullable
error) {
48.                                     if (error) {
49.                                         return;
50.                                     }
51.                                     for (BJLDocument
*document in documentArray) {
52.                                         for (NSString
*fileID in self.finishDocumentFileIDList) {
53.                                             // 如果文档已
经添加到了教室，不处理
54.                                             if
([document.fileID isEqualToString:fileID]) {
55.                                                 return;
56.                                             }
57.                                         }
58.                                         // 请求到文档信息
之后添加文档，这时认为已经添加到了教室里
59.
[self.finishDocumentFileIDList
bjl_addObject:document.fileID];
60.                                         // 转码成功后可以
获得文档的页码信息，更新本地document
61.                                         // 添加文档
62.
[self.room.documentVM
```

```
addDocument:document];
                                    }
                                }];
```

- 删除课件。

```
/**
 删除课件
 #discussion 删除成功将调用 `BJLDocumentVM` 的 `didDeleteDocument:`
 #param documentID 课件 ID
 #return BJLError:
 BJLErrorCode_invalidArguments 错误参数
 */
[self.room.documentVM deleteDocumentWithID:documentID];
```

- 监听课件变化。

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

```
// 以监听所有课件 allDocuments 的变化为例
bjl_weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room.documentVM, allDocuments)
    observer:^BOOL(NSArray<BJLDocument *> * _Nullable value, id _Nullable oldValue, BJLPropertyChange * _Nullable change) {
        bjl_strongify(self);
        // 更新数据源及相关界面控件
        self.allDocuments = [value mutableCopy];
        [self.tableView reloadData];
        [self updateViewsForDataCount];
```

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

```
// 监听添加课件的通知
[self bjl_observe:BJLMakeMethod(self.room.documentVM didAddDocument:)
        observer:^(BJLDocument *document) {
            // tableView的数据源及相关界面已经通过监听allDocuments的变化进行更新
            if(document){
                NSLog(@"document: %@ added", document);
            }
            return YES;
        }];
// 删除课件
[self bjl_observe:BJLMakeMethod(self.room.documentVM didDeleteDocument:)
           observer:^BOOL(BJLDocument *document) {
               // bjl_strongify(self);
               if(document){
                NSLog(@"document: %@ delete", document);
               }
               return YES;
           }];
```

- 监听本地课件页码。

学生翻页不会影响到远程课件翻页，如果要禁止学生本地翻页，需要在上层限制。对于单实例的文档，`BJLDocumentVM` 的 `currentSlidePage` 表示 整个教室的当前页，随 老师 / 助教

翻动课件而改变。因为学生可以回顾之前的课件，所以它不一定是本地的当前页，不能用于显示本地课件页码，SDK 限制了学生不能翻页超过远程的课件的当前页。本地课件页码通过监听 `BJLRoom` 的 `slideshowViewController` 的 `pageIndex` 获得。对于多实例的文档，只能获得本地文档的当前页 `self.documentViewController.pageIndex`，学生翻页能够超过远程课件的当前页。

```
[self
bjl_kvo:BJLMakeProperty(self.room.slideshowViewC
 pageIndex)
      observer:^BOOL(id  _Nullable value, id
_Nullable oldValue, BJLPropertyChange *
_Nullable change) {
          NSLog(@"localPage：%td", [value
integerValue]);
          return YES;
      }];
```

- 课件授权。

通过监听 `self.room.documentVM` 的 `authorizedPPTUserNumbers` 获取课件授权用户。

```
// 监听授权用户列表
[self
bjl_kvo:BJLMakeProperty(self.room.documentVM,
authorizedPPTUserNumbers)
      observer:^BOOL(id  _Nullable value, id
_Nullable oldValue, BJLPropertyChange *
_Nullable change) {
          bjl_strongify(self);
            NSLog(@"authorizedPPTUserNumbers
%@", authorizedPPTUserNumbers);
          return YES;
```

```
        }];
// 给某个用户授权
BJLError *error = [self.room.documentVM
updateStudentPPTAuthorized:authorized
userNumber:user.number];
```

- 是否禁止学生本地翻页。

通过设置 `BJLOnlineUserVM` 的 `forbidStudentChangePPT` 属性决定是否允许教室内学生本地翻页，仅限大班课类型， `BJLiveCore` **2.1.0** 版本开始支持。

- 多白板。
    - `BJLiveCore` **2.1.0** 版本开始支持多白板；
    - 大班课支持多白板功能，老师或助教可动态添加、删除白板；
    - 单页白板实例为 `BJLSlidePage` 类型， `documentID` 均为 `0` ， `slidePageIndex` 表示序号。
    - 通过 `BJLDocumentVM` 的 `addWhiteboardPage` 添加一页白板， `deleteWhiteboardPageWithIndex:` 方法删除对应 `slidePageIndex` 的白板页；
    - 通过监听 `BJLDocument` 的 `didAddWhiteboardPage:` 方法获取白板页添加通知，监听 `didDeleteWhiteboardPageWithIndex:` 方法获取白板页删除通知。

```
/**
 添加白板
 #discussion 最多同时存在 10 页白板
 #return BJLError
 */
- (nullable BJLError *)addWhiteboardPage;

/**
```

```
 添加白板通知
 #discussion 同时更新 `allDocuments`
 #param pageIndex 白板页码
 */
- (BJLObservable)didAddWhiteboardPage:(NSInteger)pageIndex;
```

```
/**
 删除白板
 #param pageIndex 白板页码，使用 BJLSlidePage 的 `slidePageIndex`
 #return BJLError
 */
- (nullable BJLError *)deleteWhiteboardPageWithIndex:(NSInteger)pageIndex;

/**
 删除白板通知
 #discussion 同时更新 `allDocuments`
 #param pageIndex 白板页码
 */
- (BJLObservable)didDeleteWhiteboardPageWithIndex:(NSInteger)pageIndex;
```

8.5 小黑板管理

- 收到小黑板发布。

```
/**
 发布小黑板的通知
 #discussion 直播间内所有用户不区分角色都会收到此信令
```

```
 #param writingBoard 小黑板信息, 此时writingBoard 的 `submitedUsers` 和 `participatedUsers` 为空,此处不做统计
 */
[self bjl_observe:BJLMakeMethod(self.room.documentVM didPublishWritingBoard:)
        filter:^BOOL(BJLWritingBoard *writingBoard) {
            return (writingBoard
                    && writingBoard.boardID.length);
        }
      observer:^BOOL(BJLWritingBoard *writingBoard) {
            bjl_strongify(self);
            // 判断是否为学生/助教
            if (writingBoard.operate == BJLWritingBoardPublishOperate_begin) {
            }
            else if (writingBoard.operate == BJLWritingBoardPublishOperate_revoke
                    && self.writingBoardViewController) {
            }
            else if (writingBoard.operate == BJLWritingBoardPublishOperate_end
                    && self.writingBoardViewController) {
            }
            return YES;
        }];
```

- 根据发布的小黑板信息创建小黑板。

```
/**
```

```
根据 writingBoard 创建对应的小黑板控制器
 #param writingBoard 小黑板信息
 #return 对应小黑板的视图控制器
 */
self.writingBoradViewController =
[room.documentVM
writingBoardViewControllerWithWritingBoard:writing
```

- 参与小黑板作答。

```
  // 参与作答
  [self.room.documentVM
participateWritingBoard:self.writingBoard.boardID];
  // 开启小黑板画笔
  [self.room.drawingVM
updateWritingBoardEnabled:status ==
BJLIcWriteBoardStatus_studentEdit];
  // 提交小黑板内容
  [self.room.documentVM
submitWritingBoard:self.writingBoard.boardID];
```

9. 画笔

老师和处于发言状态的学生可以在白板和 PPT 上添加、清除画笔，对于单文档实例，**操作画笔时用户的当前课件页面必须与老师保持一致**，对于多文档实例，操作画笔的当前页面可以和远程页面不一致。画笔管理使用 `BJLDrawingVM` 。2.x的画笔支持多种画笔模式，参考 `BJLBrushOperateMode` ，多种画笔形状，参考 `BJLDrawingShapeType` 。

9.1 画笔控制

- 显示画笔视图。

目前画笔与课件共用同一个视图 单文档：`self.room.slideshowViewController.view`，多文档：通过`BJLDocumentVM`以及`documentID`获取`-(UIViewController<BJLSlideshowUI>*)documentViewControllerWithID:(NSString*)documentID;`。因此所有能使用画笔的视图必须是文档视图。

- 开启、关闭画笔。

```
/**
 开启、关闭画笔
 #param drawingEnabled YES：开启，NO：关闭
 #return BJLError:
 #discussion BJLErrorCode_invalidCalling　错误调用，当前用户是学生、`drawingGranted` 是 NO
 #discussion 开启画笔时，单文档实例情况下如果本地页数与服务端页数不同步则无法绘制
 #discussion `drawingGranted` 是 YES 时才可以开启，`drawingGranted` 是 NO 时会被自动关闭
 */
 BJLError *error = [self.room.drawingVM updateDrawingEnabled:YES];
```

- 画笔授权。

```
/** 老师、助教: 给学生授权/取消画笔
 #param granted　　是否授权
 #param user　　授权操作的对象用户
 #return BJLError:
 BJLErrorCode_invalidUserRole　当前用户不是老师或者助教
 BJLErrorCode_invalidArguments　参数错误
 */
```

```
BJLError *error = [self.room.drawingVM updateDrawingGranted:grant userNumber:user.number];
```

```
// 所有被授权使用画笔的学生编号
@property (nonatomic, readonly, copy) NSArray<NSString *> *drawingGrantedUserNumbers;
```

- 画笔设置。

```
/**
 更新画笔操作模式

 #param operateMode 操作模式，参考 `BJLBrushOperateMode`
 #return BJLError:
 #discussion BJLErrorCode_invalidCalling drawingEnabled 是 NO
 */
BJLError *error = [self.room.drawingVM updateBrushOperateMode:operateMode];
```

```
// 切换画笔图形, 参考 `BJLDrawingShapeType`
self.room.drawingVM.drawingShapeType = BJLDrawingShapeType_rectangle; // 画矩形
```

```
/** 画笔边框颜色 */
@property (nonatomic, nonnull) NSString *strokeColor;
/** 画笔边框颜色透明度，取值范围 0~1 */
@property (nonatomic) CGFloat strokeAlpha;
/** 画笔填充颜色 */
```

```
@property (nonatomic, nullable) NSString *fillColor;
/** 画笔填充颜色透明度，取值范围 0~1, fillColor 不为空时有效 */
@property (nonatomic) CGFloat fillAlpha;
/** doodle 画笔线宽 */
@property (nonatomic) CGFloat doodleStrokeWidth;
/** 图形画笔边框线宽 */
@property (nonatomic) CGFloat shapeStrokeWidth;
/** 文字画笔字体大小 */
@property (nonatomic) CGFloat textFontSize;
/** 文字画笔是否加粗 */
@property (nonatomic) BOOL textBold;
/** 文字画笔是否为斜体 */
@property (nonatomic) BOOL textItalic;
/** 画笔开关状态，参考 `drawingGranted`、`updateDrawingEnabled:` */
@property (nonatomic, readonly) BOOL drawingEnabled;
/** doodle 画笔是否虚线 */
@property (nonatomic) BOOL isDottedLine;
/** 选中画笔时是否显示归属信息 */
@property (nonatomic) BOOL showBrushOwnerNameWhenSelected;
```

- 清空画板。

```
[self.room.slideshowViewController clearDrawing];
```

9.2 添加图片画笔

仅限**专业版小班课**。

```
/**
 添加图片画笔
 #param imageURL 图片 url
 #param relativeFrame 图片相对于画布的 frame, 各项数值取值范围为 [0.0, 1.0]
 #param documentID 目标文档 ID
 #param pageIndex   目标页
 #param isWritingBoard   是否为小黑板
 #return BJLErrorCode_invalidCalling drawingEnabled 是 NO
 */
- (nullable BJLError *)addImageShapeWithURL:(NSString *)imageURL
                                relativeFrame:(CGRect)relativeFrame
                                 toDocumentID:(NSString *)documentID
                                    pageIndex:(NSUInteger)pageIndex;
```

9.3 激光笔

仅限**专业版小班课**。

```
/**
 专业版小班课 - 激光笔位置移动请求
 #param location         激光笔目标位置
 #param documentID       激光笔所在文档的 ID
 #param pageIndex        激光笔所在文档页码
 */
- (nullable BJLError *)moveLaserPointToLocation:(CGPoint)location
                                     documentID:(nonnull NSString *)documentID
```

```
                                pageIndex:(NSUInteger)pageIndex;
```

```
/**
 专业版小班课 - 激光笔位置移动监听
 #param location        激光笔位置
 #param documentID      激光笔所在文档的 ID
 #param pageIndex       激光笔所在文档页码
 */
- (BJLObservable)didLaserPointMoveToLocation:(CGPoint)location
                                  documentID:(NSString *)documentID
                                   pageIndex:(NSUInteger)pageIndex;
```

10. 聊天

SDK 提供教室内的群聊、私聊功能（不包含显示视图），可以发送文字、图片、表情三种类型的消息，提供禁言机制。

- 聊天视图：需自行创建，SDK 提供聊天管理类型 `BJLChatVM` 。
- 显示聊天的UI界面是需要重点优化的，优化方式可以使用高度缓存，手动计算高度，不使用自动布局等方式来优化，聊天界面一直占用主线程会导致音视频和课件不同步，界面一直卡顿等问题。如果成功优化之后依旧存在卡顿，可以使用调试工具针对性能消耗较大的功能进行针对性的优化。

10.1 聊天消息监听

- 获取所有消息，2.x 之后 SDK 【不】保存聊天消息，需要上层自行维护。

```
/**
 收到历史消息 - 需要覆盖更新本地缓存的消息
 #discussion 首次进教室或断开重连会收到此回调
 #param receivedMessages 收到的所有消息
 */
[self bjl_observe:BJLMakeMethod(self.room.chatVM, receivedMessagesDidOverwrite:)
        observer:^BOOL(NSArray<BJLMessage *> *_Nullable messages) {
            bjl_strongify(self);
            if (messages.count) {
                [self.messages removeAllObjects];
                if (messages.count > 0) {
                    [self.messages addObjectsFromArray:messages];
                }
                [self.tableView bjl_clearHeightCaches];
                [self.tableView reloadData];
                [self scrollToTheEndTableView];
            }
            return YES;
        }];
```

- 监听消息增量更新。

```
[self bjl_observe:BJLMakeMethod(self.room.chatVM, didReceiveMessages:)
            observer:^BOOL(NSArray<BJLMessage *> *messages) {
                bjl_strongify(self);
                if (!messages.count) {
                    return YES;
                }
```

```
            [self.messages addObjectsFromArray:messages];
            [self.tableView reloadData];
            return YES;
        }
```

10.2 禁止聊天

```
/**
 老师: 设置全体禁言状态
 助教：设置自己所在组全组禁言
 #discussion 当前用户为老师或者大班助教设置成功后修改 `forbidAll`，否则若为分组助教设置成功后修改 `forbidMyGroup`
 #param forbidAll YES：全体或自己所在分组禁言，NO：取消全体/自己所在分组禁言
 #return BJLError:
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限
 */
[self.room.chatVM sendForbidAll:YES]; // YES:全体禁言 NO:取消全体禁言

// 判断是否处于全体禁言状态
BOOL forbidAll = self.room.chatVM.forbidAll;
```

```
/**
 老师: 对某人禁言
 #discussion `duration` 为禁言时间
 #param user     禁言对象
 #param duration 禁言时长
 #return BJLError:
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限
```

```
 */
[self.room.chatVM sendForbidUser:user
duration:60.0];

// 判断用户是否被禁言
BOOL forbidMe = self.room.chatVM.forbidMe;
```

```
// 监听用户被禁言通知
[self
bjl_observe:BJLMakeMethod(self.room.chatVM,
didReceiveForbidUser:fromUser:duration:)
        observer:^BOOL(BJLUser *forbidUser,
BJLUser *fromUser, NSTimeInterval duration){
            NSLog(@"%@被%@禁言%f秒",
forbidUser.name,fromUser.name,duration);
            return YES;
        }];
```

10.3 发送、接收消息

- 发送消息：发送前需判断用户是否被禁言。

```
// 发送文字消息
[self.room.chatVM
sendMessage:self.textField.text];
```

```
// 发送图片消息：image 为需要发送的图片，fileURL
为它的文件路径
bjl_weakify(self);
// 上传图片
[self.room.chatVM uploadImageFile:fileURL
                        progress:^(CGFloat progress){
                            bjl_strongify(self);
                            // 显示进度
```

```
self.imageUploadingView.progress = progress;
                    }
                        finish:^(NSString * _Nullable imageURLString, BJLError * _Nullable error) {
                            bjl_strongify(self)
                            if(imageURLString){
                                NSDictionary *imageData = [BJLMessage messageDataWithImageURLString:imageURLString imageSize:image.size];
                                [self.room.chatVM sendMessageData:imageData];
                            }
                            else{
                                NSLog(@"error:%@", error);
                            }
                        }];
```

```
// 发送表情
// 需要在教室内才可以获取到表情，获取 emotion 数组
NSArray<BJLEmoticon *> *emoticons = [BJLEmoticon allEmoticons];
if (emoticons.count > 0) {
    // 模拟表情选择，这里直接选择第一个表情
    BJLEmoticon *emoticon = [emoticons objectAtIndex:0];
    if (emoticon) {
        //发送表情
        [self.room.chatVM sendMessageData:[BJLMessage messageDataWithEmoticonKey:emoticon.key]];
    }
}
```

- 接收消息。

```
// 接收表情和文字混排的消息，获取NSAttributedString 显示即可
self.messageLabel.attributedText = [message attributedEmoticonStringWithEmoticonSize:fontSize + 4.0 attributes:@{NSFontAttributeName:[UIFont systemFontOfSize:fontSize],NSForegroundColorAttri(horizontal ? [UIColor whiteColor] : [UIColor bjp_darkGrayTextColor])} cached:YES cachedKey:(horizontal ? @"white" : @"gray")];
```

```
// 接收文字和图片参考发送消息
// 接收表情消息 type == BJLMessageType_emoticon, 优先判断cachedImage
if (self.message.type == BJLMessageType_emoticon) {
    if (self.message.emoticon.cachedImage) {
        self.emoticonImageView.image = self.message.emoticon.cachedImage;
    }
    else {
        bjl_weakify(self);
        [self.emoticonImageView bjl_setImageWithURL:self.message.emoticon.urlStri
                                        placeholder:nil

completion:^(UIImage * _Nullable image, NSError * _Nullable error, NSURL * _Nullable imageURL) {
            bjl_strongify(self);
            if (image) {
                self.message.emoticon.cachedImage = image;
            }
```

```
        }];
    }
}
```

- 私聊、频道聊天。

```
/**
 指定对象，发送文字消息
 #param text 文字消息内容
 #param user 发送对象
 #return BJLError:
 BJLErrorCode_invalidArguments  参数错误；
 BJLErrorCode_invalidCalling    错误调用，如禁言状态时调用此方法发送消息。
 */
[self.room.chatVM sendMessage:message toUser:targetUser];

/**
 指定频道，发送文字消息
 #discussion 最多 BJLTextMaxLength_chat 个字符
 #discussion 成功后会收到消息通知
 #discussion 学生在禁言状态不能发送消息，参考 `forbidMe`、`forbidAll`
 #discussion 参考 `BJLMessage`
 #param text 消息，不能是空字符串或 nil
 #param channel 频道
 */
[self.room.chatVM sendMessage:message channel:@"小组频道"];
```

10.4 置顶聊天

将聊天消息置顶，取消置顶。

```objc
/** 聊天的置顶消息 */
@property (nonatomic, readonly, nullable) BJLMessage *stickyMessage;
/**
 老师: 设置置顶消息
 #discussion message 为空表示取消置顶内容, 不为空表示设置新的置顶消息
 #discussion 设置成功后修改 `stickyMessage`
 #return BJLError:
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限。
 */
BJLError *error = [self.room.chatVM sendStickyMessage:message];
```

10.5 聊天翻译

支持聊天消息的英汉互译。

```objc
/**
 聊天消息英汉互译，每秒最多仅允许发送一条
 #param message 消息
 #param messageUUID 用于区分消息的唯一ID,不建议使用BJLMessage.ID, 不保证其唯一性
 #param translateType 英->汉 or 汉->英
 */
BJLError *error = [self.room.chatVM translateMessage:message
                                         messageUUID:messageUUID
                                       translateType:([self isEN:message] ? BJLMessageTranslateTypeZHtoEN : BJLMessageTranslateTypeENtoZH)];

```

```
/**
 收到翻译结果
 #param translation 消息翻译结果
 #param messageUUID 消息唯一ID
 */
[self bjl_observe:BJLMakeMethod(self.room.chatVM, didReceiveMessageTranslation:messageUUID:)
        observer:^BOOL(NSString *translate, NSString *messageUUID) {
            bjl_strongify(self);
            [self didReceiveMessageTranslation:translate messageUUID:messageUUID];
            return YES;
        }];
```

11. 录课

录课是将当前教室的情景、信息以及互动记录录制到云端生成回放，通过回放功能可以再现教室的情景。本 SDK 不包含回放功能，如需集成请参考 iOS 点播回放 Core SDK。

录课管理类型为 `BJLServerRecordingVM`，实例 `serverRecordingVM` 在创建教室时被初始化。

- 获取云端录课状态。

```
// 云端录课状态，反映当前云端录课是否开启
BOOL serverRecording = self.room.serverRecordingVM.serverRecording;

/* 云端录制详细状态
    BJLServerRecordingState_ready,                  // 未开启云端录制
```

```
    BJLServerRecordingState_recording,
// 开启过云端录制并且未转码或者正在云端录制中，可继续录制。如果现在在录制中，长期课可以停止录制，请求转码后开启新的录制
    BJLServerRecordingState_transcoding,
// 云端录制转码中，只能开启新的云端录制，不能继续录制。短期课不会有这个状态
    BJLServerRecordingState_disable,
// 云端录制不可用，是短期课已经录制过。长期课不会有这个状态
*/
@property (nonatomic, readonly)
BJLServerRecordingState state;
```

- 老师开启/停止云端录课：上课状态才能开启录课，参考 `roomVM.liveStarted`，此方法需要发起网络请求、检查云端录课是否可用。

```
[self.room.serverRecordingVM
requestServerRecording:YES]; // YES:开启, NO:关闭
```

```
/**
 请求当前云端录制详细状态

 @param completion 状态更新完成，可以根据状态来进行云端录制
 */
- (void)requestServerRecordingState:(void (^
__nullable)(void))completion;
```

- 请求立即转码回放。

```
/**
 请求立刻转码回放
```

```
 #return BJLError:
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限。
 */
 BJLError *error = [self.room.serverRecordingVM requestServerRecordingTranscode];

 // 转码回放请求被接受
[self bjl_observe:BJLMakeMethod(self.room.serverRecord requestServerRecordingTranscodeAccept)
            observer:^BOOL{
                // bjl_strongify(self);
                return NO;
            }];
```

- 监听云端录课不可用的通知。

```
[self bjl_observe:BJLMakeMethod(self.room.serverRecord requestServerRecordingDidFailed:)
        observer:^BOOL(NSString *message) {
            NSLog(@"request server recording failed:%@", message);
            return YES;
        }];
```

12. 教室工具

`BJLRoomVM` 用于记录教室的信息、状态，以及教室的一些工具、功能。可以根据实际需要选取使用。

12.1 教室上下课

可以监听上课时间，进入教室时间，上下课状态等信息。

```objc
/** 上课状态 */
@property (nonatomic, readonly) BOOL liveStarted;
/** 实际上课开始时间 */
@property (nonatomic, readonly) NSTimeInterval classStartTimeMillisecond;
/** 理论排课开始时间 */
@property (nonatomic, readonly) NSTimeInterval classStartTimesecond;
/** 进入教室时间 */
@property (nonatomic, readonly) NSTimeInterval enteringTimeInterval; // seconds since 1970
```

12.2 助教权限

可以获取助教拥有的教室权限，助教的权限可以由老师控制。

```objc
// example: 监听权限变更，获取当前是否拥有上课权限
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveAssistantaAuthorityChanged) observer:^BOOL{
    bjl_strongify(self);
    BOOL enableLiveStart = [self.room.roomVM getAssistantaAuthorityWithClassStartEnd];
    return YES;
}];
```

12.3 公告

用户可以查看教室内由老师发布的公告，公告可包含跳转链接。 `BJLRoomVM` 提供公告的获取、发布方法，也可以监听公告变化，即时更新。

- 获取教室公告。

```
// 获取教室公告，连接教室后、掉线重新连接后自动调用 loadNotice，获取成功后修改 notice
[self.room.roomVM loadNotice];
BJLNotice *notice = self.room.roomVM.notice;
```

- 监听公告变化，即时显示。

```
bjl_weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room.roomVM, notice)
    observer:^BOOL(BJLNotice * _Nullable notice, id _Nullable oldValue, BJLPropertyChange * _Nullable change) {
        bjl_strongify(self);
        self.noticeTextView.text = notice.noticeText.length ? notice.noticeText : nil;
        return YES;
    }];
```

- 发布公告。

```
// noticeText:公告内容 linkURL:公告跳转链接
[self.room.roomVM sendNoticeWithText:noticeText linkURL:noticeURL];
```

12.4 跑马灯

跑马灯是直播间内漂浮的文字信息，可以通过后台自定义样式，内容。

```
/** 跑马灯内容 <包括文本内容，文本颜色，字体大小，透明度> */
```

```
@property (nonatomic, readonly, copy, nullable) BJLLamp *lamp;
/** 跑马灯内容 */
@property (nonatomic, readonly, copy, nullable) NSString *lampContent;
```

12.5 课后评价

课后评价是课程结束时，一个对课堂情况等的问卷反馈。目前课后评价的使用嵌入的 webview 实现，可以通过 `evaluationRequest` 获取当前教室课后评价的地址展示，具体展示方式参考 `EvaluationViewController` 。支持后台配置课后评价的内容。

```
// 获取课后评价，webview 加载请求
self.request = [self.room.roomVM evaluationRequest];
[self.webView loadRequest:self.request];
```

12.6 点名

点名由老师发布，学生接收并答到。SDK 不支持发起点名。

- 收到点名。

```
/**
 学生: 收到点名
 #discussion 学生需要在规定时间内 `timeout` 答到 - 调用 `answerToRollcall`
 #discussion 参考 `rollcallTimeRemaining`
 #param timeout 超时时间
 */
- (BJLObservable)didReceiveRollcallWithTimeout:(NSTimeInterval)timeout;
```

- 点名倒计时。

```
/**
 点名倒计时
 #discussion 每秒更新
 */
@property (nonatomic, readonly) NSTimeInterval rollcallTimeRemaining;
```

- 收到点名取消。

```
/**
 学生: 收到点名取消
 #discussion 可能是老师取消、或者倒计时结束
 #discussion 参考 `rollcallTimeRemaining`
 */
- (BJLObservable)rollcallDidFinish;
```

- 答到。

```
/**
 学生: 答到
 #return BJLError:
 BJLErrorCode_invalidCalling    错误调用，如老师没有点名或者点名已过期;
 BJLErrorCode_invalidUserRole    错误权限，要求非试听学生权限。
 */
- (nullable BJLError *)answerToRollcall;
```

12.7 专注度检测

SDK 会自动进行专注度检测，在后台或者 APP 处于不活跃的状态都将会认为是不专注的，老师提醒当前用户不专注时，在 APP

重新处于活跃状态时可以收到老师的提醒。

```
    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveAttentionWarning:)
            observer:^BOOL(NSString *content) {
        bjl_strongify(self);
        [self showProgressHUDWithText:content];
        return YES;
    }];
```

12.8 点赞

- 助教和老师可以给学生点赞，学生无法点赞，助教和老师不能被点赞，下课【不】清空点赞记录。

```
/**
 点赞数据
 #discussion key --> userNumber
 #discussion value --> 点赞数
 */
@property (nonatomic, readonly, nullable) NSDictionary<NSString *, NSNumber *> *likeList;
@property (nonatomic, readonly, nullable) NSDictionary<NSNumber *, NSNumber *> *grouplikeList;
```

- 点赞用户。

```
/**
 点赞
 #param userNumber userNumber
 #return error:
```

```
 BJLErrorCode_invalidCalling    错误调用，如用户不在线；
 BJLErrorCode_invalidUserRole    错误权限，如点赞用户不能是学生，被点赞用户不能是老师，助教。
 */
BJLError *error = [self.room.roomVM sendLikeForUserNumber:userNumber];
```

- 收到点赞。

```
/**
 收到个人点赞
 #discussion 收到的所有点赞都在点赞记录中，包括本次收到的点赞
 #param userNumber userNumber
 #param records 点赞记录 key --> userNumber，value --> 点赞数
 */
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveLikeForUserNumber:records:)
        observer:^BOOL(NSString *userNumber, NSDictionary<NSString *, NSNumber *> *records) {
                return YES;
        }];
/**
 收到一次分组/全员点赞
 #discussion groupID 为 0 表示全员点赞,非0表示分组点赞，非 0 时 groupName 未分组名
 #param groupid=0 时， 为全员点赞， 不计入个人/分组点赞数量
 */
[self bjl_observe:BJLMakeMethod(self.room.roomVM,
```

```
didReceiveLikeForGroupID:groupName:)
observer:^BOOL(NSInteger groupID, NSString
*groupName) {
        bjl_strongify(self);
        if (groupID != 0 && self.user.groupID ==
groupID) {
            [self updateLikeCount];
        }
        return YES;
}];
```

12.9 红包雨

- 创建红包雨。

```
/**
 创建红包雨
 #param amount 红包总数
 #param score 学分总数
 #param duration 红包雨时长
 #param completion 红包雨活动ID
 #return task
 */
- (nullable NSURLSessionDataTask
*)createEnvelopeRainWithAmount:
(NSInteger)amount score:(NSInteger)score
duration:(NSInteger)duration completion:
(nullable void (^)(NSInteger envelopeID, BJLError
* _Nullable error))completion;
```

- 抢红包。

```
/**
 抢红包
```

```
3. #discussion 活动结束时，completion 也返回 0，并且没有 task
4. #param envelopeID 红包雨活动ID
5. #param completion 抢到的学分
6. #return task
7. */
8. - (nullable NSURLSessionDataTask *)grapEnvelopeWithID:(NSInteger)envelopeID completion:(nullable void (^)(NSInteger score, BJLError * _Nullable error))completion;
```

- 获取学生抢到的学分总数。

```
1. /**
2. 获取学生抢到的学分总数
3. #param userNumber user number
4. #param completion 学分总数
5. #return task
6. */
7. - (nullable NSURLSessionDataTask *)requestTotalScoreWithUserNumber:(NSString *)userNumber completion:(nullable void (^)(NSInteger totalScore, BJLError * _Nullable error))completion;
```

- 获取指定红包雨活动的最终结果。

```
1. /**
2. 获取指定红包雨活动的最终结果
3. #param envelopeID 红包雨活动ID
4. #param completion completion 红包雨活动结果
5. #return task
6. */
7. - (nullable NSURLSessionDataTask *)requestResultWithEnvelopeID:
```

```
(NSInteger)envelopeID completion:(nullable void
(^)(BJLEnvelopeResult * _Nullable result, BJLError
* _Nullable error))completion;
```

- 获取指定红包雨活动的排行榜。

```
/**
 获取指定红包雨活动的排行榜
 #param envelopeID envelopeID
 #param completion completion 排行榜数据
 #return task
 */
- (nullable NSURLSessionDataTask
*)requestRankListWithEnvelopeID:
(NSInteger)envelopeID completion:(nullable void
(^)(NSArray<BJLEnvelopeRank *> * _Nullable,
BJLError * _Nullable))completion;
```

- 开始红包雨。

```
/**
 开始红包雨
 #param envelopeID 红包雨活动ID
 #param duration 红包雨时长
 #return error
 */
- (nullable BJLError *)startEnvelopRainWithID:
(NSInteger)envelopeID duration:
(NSInteger)duration;
```

- 收到红包雨。

```
/**
 收到红包雨
```

```
 #param envelopeID 红包雨活动ID
 #param duration 红包雨时长
 */
- (BJLObservable)didStartEnvelopRainWithID:
(NSInteger)envelopeID duration:
(NSInteger)duration;
```

12.10 测验

测验由老师发布，学生接收并答题。题目类型为 `BJLSurvey`，每个题目有序号和多个 `BJLSurveyOption` 类型的选项，`BJLSurveyOption` 的 `key` 标识一个选项，`value` 则代表选项的具体内容。SDK 不支持发布测验，相关的 API 在 BJLRoomVM 中。低于**2.1.0** 版本的SDK支持的测验是非 H5 页面的测验，即旧版测验，新版测验需要参考 UI SDK 的 `BJLQuizWebViewController` 实现。**2.1.0** 版本开始支持新版测验的 native 集成方式。

12.10.1 新版测验 native 集成方式

参考 `BJLRoomVM` 的 `测验 V2 native 方式` 部分。

- 测验状态列表。

```
/**
 当前的新版测验状态列表
 #discussion key -> 测验 ID，value -> 测验状态（参考 `BJLQuizState`），测验的先后根据测验 ID 从小到大对应
 */
@property (nonatomic, readonly, nullable)
NSDictionary<NSString *, NSNumber *>
*quizStateList;
```

- 当前正在进行的测验 ID。

```
/**
 正在进行的测验
 #discussion 至多只有一个测验正在进行，如果没有正在进行的测验，值为空
 */
@property (nonatomic, readonly, nullable) NSString *currentQuizID;
```

- 加载测验列表请求。

```
/**
 加载测验列表
 #param completion quizList 仅 quiz ID，title，state 可用
 #return task
 */
[self.room.roomVM loadQuizListWithCompletion:^(NSArray<BJLQuiz *> * _Nullable quizList, BJLError * _Nullable error) {
    // bjl_strongify(self);
    // your code
 }];
```

- 老师、助教：新增或更新测验。

```
/**
 新建，更新测验，老师或助教身份
 #discussion 创建新测验，测验 ID 使用 0，否则更新测验
 #param quiz 测验内容，参考 `BJLQuiz`
 #param completion 测验 ID
 #return task
 */
```

```
[self.room.roomVM updateQuiz:myQuiz
                  completion:^(NSString * _Nullable quizID, BJLError * _Nullable error) {
                      // bjl_strongify(self);
                      // your code
                  }];
```

- 老师、助教：删除测验。

```
/**
 删除测验，老师或助教身份
 #param quizID 测验 ID
 #param completion 测验 ID
 #return task
 */
[self.room.roomVM deleteQuizWithID:targetQuizID
                        completion:^(NSString * _Nullable quizID, BJLError * _Nullable error) {
                            // bjl_strongify(self);
                            // your code
                        }];
```

- 加载测验的详细内容。

```
/**
 加载测验详细内容
 #discussion 所有角色，对于老师或助教，如果测验结束了，有答题情况了，会返回测验的答题情况，对于学生，不会返回答题情况
 #param quizID 测验 ID
 #param completion BJLQuiz，测验详细信息
 #return task
 */
```

```
- (nullable NSURLSessionDataTask *)loadQuizDetailWithID:(NSString *)quizID completion:(nullable void (^)(BJLQuiz * _Nullable quiz, BJLError * _Nullable error))completion;
```

- 开始测验。

```
/**
 开始测验，老师或助教身份
 #param quizID quizID
 #param force 是否强制参加
 #return BJLError
 */
- (nullable BJLError *)startQuizWithID:(NSString *)quizID force:(BOOL)force;

// 监听测验开始
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didStartQuizWithID:force:)
        observer:(BJLMethodObserver)^BOOL(NSString *quizID, BOOL force) {
            // bjl_strongify(self);
            NSLog(@"测验%@开始，强制参加：%@", quizID, force? @"YES" : @"NO");
            return YES;
        }];
```

- 结束测验。

```
/**
 老师、助教：结束测验
 #param quizID 测验 ID
 #return BJLError
```

```
 */
- (nullable BJLError *)endQuizWithID:(NSString *)quizID;

// 监听测验结束
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didEndQuizWithID:)
        observer:^BOOL(NSString *quizID) {
            // bjl_strongify(self);
            NSLog(@"测验%@结束", quizID);
            return YES;
        }];
```

- 发布测验答案。

```
/**
 发布测验答案，老师或助教身份
 #param quiz 需要发布答案的测验 ID
 #return BJLError
 */
- (nullable BJLError *)publishQuizSolutionWithID:(NSString *)quizID;

// 发布测验答案通知，可通过 KVO 监听
- (BJLObservable)didReceiveQuizWithID:(NSString *)quizID solution:(NSDictionary<NSString *, id> *)solutions;
```

- 加载当前测验。

```
/**
 加载当前测验
 #discussion 学生身份如果回答过，将返回回答的结果
 #return BJLError
```

```
 */
- (nullable BJLError *)loadCurrentQuiz;

// 测验加载回调，学生用户 如果回答过，将返回回答的结果
- (BJLObservable)didLoadCurrentQuiz:(BJLQuiz *)quiz;
```

- 提交测验。

```
/**
 提交测验，学生身份，此方法无提交完成的回调
 #param quizID 测验 ID
 #param solutions 测验回答 key -> 问题 ID，value -> 问题回答，对于 value，Radio 类型的问题的值为选项 ID，Checkbox 类型的问题的值为选项 ID 数组，ShortAnswer 类型的问题值为简答的关键内容
 #return BJLError
 */
- (nullable BJLError *)submitQuizWithID:(NSString *)quizID solution:(NSDictionary<NSString *, id> *)solutions;
```

```
/**
 收到学生提交测验，老师或助教身份
 #param quizID 测验ID
 #param solutions 学生答题情况
 */
- (BJLObservable)didSubmitQuizWithID:(NSString *)quizID solution:(NSDictionary<NSString *, id> *)solutions;
```

- 大小班场景测验。

```
/**
 位于小班，加载大班的已经结束的测验列表
 @return BJLError
 */
- (nullable BJLError *)loadParentRoomFinishedQuizList;

/**
 位于小班，收到大班结束的测验信息
 @param quizList 测验列表
 */
- (BJLObservable)didLoadParentRoomFinishedQuizList:(nullable NSArray<BJLQuiz *> *)quizList;
```

12.10.2 旧版测验

不推荐使用，如有需要可联系百家云后台开启。

- 请求历史题目。

```
[self.room.roomVM loadSurveyHistory];
```

- 监听收到历史题目以及当前用户的答题情况。

```
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveSurveyHistory:rightCount:wrongCount:)
        observer:^BOOL(NSArray<BJLSurvey *> *surveyHistory, NSInteger rightCount, NSInteger wrongCount) {
            NSLog(@"receive %td history surveys, %td are right, %td are wrong", surveyHistory.count, rightCount, wrongCount);
            return YES;
```

```
5.          }];
```

- 学生：收到新题目。

```
1. [self
   bjl_observe:BJLMakeMethod(self.room.roomVM,
   didReceiveSurvey:)
2.          observer:^BOOL(BJLSurvey *survey) {
3.              NSLog(@"did receive survey: %@",
   survey.question);
4.              return YES;
5.          }];
```

- 学生答题：支持多选。

```
1. /**
2.  @param answers  学生选择的 BJLSurveyOption 的
   key 的数组，
3.  @param result   与每个 BJLSurveyOption 的
   isAnswer 比对得出，如果一个题目下所有
   BJLSurveyOption 的 isAnswer 都是 NO 表示此题目没
   有标准答案
4.  @param order    序号，BJLSurvey 的 order
5.  */
6. [self.room.roomVM sendSurveyAnswers:answers
   result:result order:order];
```

```
1. // 例：假设当前题目为 survey，学生选择的
   BJLSurveyOption 的 key 的数组为 selectAnswers
2. NSMutableArray *answers;
3. BJLSurveyResult result;
4. // 筛选出当前题目的所有正确答案
5. for (BJLSurveyOption *option in survey.options) {
6.     if (option.isAnswer) {
```

```
            [answers addObject:option.key];
        }
    }

    if (answers.count <= 0) {
        // 无标准答案
        result = BJLSurveyResultNA;
    }
    else if ([selectAnswers isEqualToArray:answers])
    {
       // 所选答案与正确答案匹配
        result = BJLSurveyResultRight;
    }
    else {
        result = BJLSurveyResultWrong;
    }

     // 发送答案
    [self.room.roomVM
    sendSurveyAnswers:selectAnswers result:result
    order:survey.order];
```

- 监听收到答题统计。

```
    /**
     @param results 统计结果，这个 NSDictionary 的 key-value 分别是 BJLSurveyOption 的 key 和选择该选项的人数
     @param order   序号，BJLSurvey 的 order
    */
    [self
    bjl_observe:BJLMakeMethod(self.room.roomVM,
    didReceiveSurveyResults:order:)
            observer:^BOOL(NSDictionary<NSString *,
    NSNumber *> *results, NSInteger order) {
```

```
        NSLog(@"did receive results  of survey: %td", order);
        return YES;
    }];
```

- 学生：收到答题结束。

```
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didFinishSurvey:)
    observer:^BOOL {
        return YES;
    }];
```

12.11 问答

问答是类似于聊天的互动功能，学生提出问题，老师或者助教进行回答，问答如果在未发布状态是只有提问者和老师、助教可见，在发布状态所有人可见。问答通过分页加载。

- 获取问答数据。

```
/**
 加载指定页码的问答
 #param page 页码，从0开始计数
 #param count 每一页最多返回5条，如果数据较多，需要分页请求
 #return error
 */
BJLError *error = [self.room.roomVM loadQuestionHistoryWithPage:self.currentQuestionP
 countPerPage:perPageQuestionCount];

/**
 收到指定页码的问答数据
```

```
 #param history 问答数据，可能为空列表
 #param currentPage 当前页码，从0开始计数
 #param totalPage 最大页码，从0开始计数
 */
[self
bjl_observe:BJLMakeMethod(self.room.roomVM,
didLoadQuestionHistory:currentPage:totalPage:)
          observer:^BOOL(NSArray<BJLQuestion
*> *history, NSInteger currentPage, NSInteger
totalPage) {
              bjl_strongify(self);
              [self.tableView reloadData];
              return YES;
          }];
```

- 创建、发布、取消发布、回复问答。

```
/**
 创建问答
 #param question 问题内容
 */
- (BJLError *)sendQuestion:(NSString *)question;
/**
 创建问答成功
 #param question 问答，包括问答 ID
 */
- (BJLObservable)didSendQuestion:(BJLQuestion
*)question;

/**
 发布问答，需要先成功创建问答
 #param questionID 问答 ID
 */
- (BJLError *)publishQuestionWithQuestionID:
(NSString *)questionID;
```

```
17.
18. /**
19.  取消发布问答
20.  #param questionID 问答 ID
21.  */
22. - (BJLError *)unpublishQuestionWithQuestionID:
    (NSString *)questionID;
23.
24. /**
25.  回复问答
26.  #param questionID 问答 ID
27.  #param reply 回复内容
28.  */
29. - (BJLError *)replyQuestionWithQuestionID:
    (NSString *)questionID reply:(NSString *)reply;
```

- 改变用户问答状态。

```
1. /**
2.  禁止提出问答的 userNumber 列表
3.  #discussion 列表包含禁止提出问答的 userNumber
4.  */
5. @property (nonatomic, readonly, nullable)
   NSSet<NSString *> *forbidQuestionList;
6.
7. /**
8.  改变问答状态
9.  #param user user
10.  #param forbid 是否禁止问答
11.  */
12. - (BJLError *)switchQuestionForbidForUser:
    (BJLUser *)user forbid:(BOOL)forbid;
```

12.12 网页

网页是用于老师发布给学生打开一个指定 URL 地址的网页，可以用于各种场景。

```
/**
 更新网页信息
 #param urlString 网址
 #param open YES：打开，NO：关闭
 #return BJLError
 */
[self.room.roomVM updateWebPageWithURLString:urlString open:publish];

/**
 收到网页信息更新
 #param urlString 网址
 #param open YES：打开，NO：关闭
 #param isCache 是否是缓存
 */
[self bjl_observe:BJLMakeMethod(self.room.roomVM, didUpdateWebPageWithURLString:open:isCache:)
        observer:(BJLMethodObserver)^BOOL(NSString *urlString, BOOL open, BOOL isCache) {
            bjl_strongify(self);
            if (open) {
            // open
            }
            else {
                // close
            }
            return YES;
}];
```

12.13 学情报告

学情报告目前主要包括表情报告，是通过 AI 分析上课时用户的特征表情数据，截取生成统计报告。目前仅 **webrtc** 教室支持。

- 请求生成学情报告。

```
[self.room.roomVM generateExpressReportWithCompletion:^(NSString * _Nullable ID, BJLError * _Nullable error) {
    bjl_strongify(self);
    if (error) {
     // error
    }
    else {
    // wait and get report
    }
}];
```

- 学情报告生成状态。

```
/**
 获取学情报告进程状态
 #param completion 进程状态，参考 `BJLTaskStatus`
 #return error
 */
[self.room.roomVM requestExpressReportProgressWithCompletion:^(BJ status, BJLError * _Nullable error) {
        bjl_strongify(self);
        switch (status) {
            case BJLTaskStatus_unknow:
            case BJLTaskStatus_failed:
            case BJLTaskStatus_timeOut:
```

```
            break;

        case BJLTaskStatus_processing:
            break;

        case BJLTaskStatus_finished:
            break;

        default:
            break;
    }
}];
```

- 加载学情报告。

```
/**
 课后学情报告
 #param userNumber 用户 number
 #return NSURLRequest
 */
self.request = [self.room.roomVM expressReportRequestWithUserNumber:usrNumber
[self.webView loadRequest:self.request];
```

12.14 计时器

- 发布计时器。

```
/**
 发布计时器
 #param totalTime 计时总时长，单位 秒
 #param countDownTime 当前计时剩余计时时长，单位 秒
 #param isDecrease 是否为倒计时，NO表示正计时
 #return BJLError
```

```
 */
BJLError *error = [self.room.roomVM
requestPublishTimerWithTotalTime:self.totalTime

countDownTime:leftCountDownTime

isDecrease:self.isDecrease];
```

- 收到计时器。

```
/**
 收到计时器信息更新
 #param time 倒计时时间
 #param open YES：发布，NO：关闭
 */
[self
bjl_observe:BJLMakeMethod(self.room.roomVM,
didReceiveTimerWithTotalTime:countDownTime:isD


        observer:(BJLMethodObserver)^BOOL
(NSInteger totalTime, NSInteger countDownTime,
BOOL isDecrease){
        bjl_strongify(self);
         return YES;
}];
```

- 暂停、取消计时器。

```
// 暂停计时器
BJLError *error = [self.room.roomVM
requestPauseTimer];
// 取消计时器
BJLError *error = [self.room.roomVM
requestStopTimer];
```

12.15 抢答器

- 发布抢答器。

```
/**
 请求发布抢答器
 #param time 抢答器倒计时
 #return BJLError
 */
BJLError *error = [self.room.roomVM requestPublishQuestionResponderWithTime:time];

// 收到抢答器开始信息
-(BJLObservable)didReceiveQuestionResponderWithT(NSInteger)time;
```

- 学生抢答。

```
BJLError *error = [self.room.roomVM submitQuestionResponder];
```

- 结束、撤销抢答器。

```
// 撤销抢答器
BJLError *error = [self.room.roomVM requestRevokeQuestionResponder];
// 关闭抢答器窗口
BJLError *error = [self.room.roomVM requestCloseQuestionResponder];
```

12.16 答题器

答题由老师发布，学生接收并答题。SDK 支持发布答题，相关的 API 在 BJLRoomVM 中。

- 收到答题。

```
    // 答题开始
    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveQuestionAnswerSheet:)
            observer:^BOOL(BJLAnswerSheet *answerSheet) {
            bjl_strongify(self);
            [self showProgressHUDWithText:@"答题开始"];
            [self showAnswerSheet];
            return YES;
        }];
```

- 答题结束。

```
    // 答题结束
    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveEndQuestionAnswerWithEndTime:)
            observer:(BJLMethodObserver)^BOOL(NSTimeInterval endTimeInterval) {
            bjl_strongify(self);
            [self showProgressHUDWithText:@"答题已结束"];
            [self clearAnswerSheet];
            return YES;
        }];
```

- 提交答案。

```
/**
提交答案
 #param answerSheet 答题表：options 数组中的 BJLAnswerSheetOption 实例对应各个选项, 它的 seletced 属性表示该选项是否被选中
 */
 BJLError *error = [self.room.roomVM submitQuestionAnswer:answerSheet];
```

12.17 抽奖

抽奖支持标准抽奖和口令抽奖二种抽奖方式。标准抽奖是在一系列用户中抽出中奖用户，口令抽奖是在老师发布指定口令，对在限定时间内发出口令的用户抽奖，SDK 支持参与和接收抽奖结果。

- 收到抽奖结果。

```
    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveLotteryResult:) observer:^BOOL(BJLLottery *lottery){
        bjl_strongify(self);
        if (BJLLotteryType_Command == lottery.type) {
            // 口令抽奖
        }
        else if (BJLLotteryType_Standard == lottery.type) {
            // 标准抽奖
        }
        return YES;
    }];
```

- 收到口令抽奖开始。

```
    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveBeginCommandLottery:) observer:^BOOL{
        bjl_strongify(self);
        // 收到口令抽奖开始
        return YES;
    }];
```

- 参与口令抽奖。

在聊天区发出了和要求口令相同的聊天内容后，调用该方法去参与口令抽奖。

```
    [self.room.roomVM requestHitCommandLottery];

    [self bjl_observe:BJLMakeMethod(self.room.roomVM, didReceiveHitCommandLottery:) observer:^BOOL{
        bjl_strongify(self);
        // 收到参与口令抽奖成功回调
        return YES;
    }];
```

- 提交中奖信息。

中奖的用户可以填入联系方式，方便后期联系该用户。

```
/**
 提交中奖信息
 #param userName 用户名
 #param mobile 用户联系电话
 #param beginTime 抽奖开始时间
```

```
 #param completion 提交完成的回调
 */
- (void)submitLotteryUserName:(NSString *)userName
                       mobile:(NSString *)mobile
                    beginTime:(NSTimeInterval)beginTime
                   completion:(nullable void (^)(BOOL success))completion;
```

13. 自习室

自习室是以小班课教室为基础的具有在线静默自习、直播讨论、互动辅导的在线教室。学生可以通过自习室和小伙伴在线一起学，还可以求助辅导老师。而老师可以切换到辅导模式来进行详细辅导。

自习室主体逻辑在 `BJLStudyRoomVM`，实例 `BJLStudyRoomVM` 在创建教室时被初始化。

- 自习室模式切换

```
/** 自习室模式切换 */
- (nullable BJLError *)studyRoomSwitchToMode:(BJLIcStudyRoomModeType)mode;
```

13.1 获取自习室公告、时长排行榜、学生列表

- 自习室公告

```
/**
 获取自习室公告
 #param completion title:公告标题  content:公告内容
 #return task
```

```
 */
- (nullable NSURLSessionDataTask *)getStudyRoomTipsWithCompletion:(nullable void (^)(NSString * _Nullable title, NSString * _Nullable content, BJLError * _Nullable error))completion;
```

- 时长排行榜

```
/** 请求自习室自习时间排行榜数据 */
- (nullable BJLError *)requestStudyRoomTimeRankList;
```

- 学生列表

```
/** 请求自习室台上用户列表数据, 目前仅仅小班课自习室可用 */
- (nullable BJLError *)requestStudyRoomActiveUserList;
```

集成常见问题

1. Block 监听相关问题

1.1 监听不到对象的 属性变化 / 方法调用

解决方法：

- 检查添加监听时监听对象是否为空：SDK 中。
- 检查 `filter` 中过滤条件是否正确。
- 检查 `observer` 中是否 `return NO` 导致监听取消。

2. 音视频相关问题

2.1 音视频用户列表为空

准备播放视频时，获取的 `self.room.playingVM.playingUsers` 为空。

解决方法：

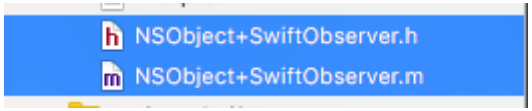
- `playingUsers` 是随时变化的，不能用直接取值的方法来获取音视频列表，应该监听 `self.room.playingVM` 的 `playingUsers` 属性的变化，即时获取最新列表，参考监听音视频用户列表。使用监听方式出现此问题则请参考后续部分。
- 检查教室内是否有用户在发言（打开了音频或视频）。

3. Swift 项目集成 SDK 相关问题

3.1 使用 Block 方式监听方法调用时无回调

解决方法：

- 在工程中添加如下适配文件：



- 在 `Bridging Header` 中导入适配文件。

```
#import "NSObject+SwiftObserver.h"
```

- 使用适配文件提供的监听方法进行监听，以监听 `BJLRoom` 的 `enterRoomSuccess` 为例：

```
self.bjl_observeEnterRoomSuccess(forTarget:
self.room,
                                 filter: { () -> Bool in
```

```
                return true },
            observer: { () -> Bool in
                // your code
                return true
            })
```

3.2 属性监听

Swift 使用 Block 的方式监听属性时必须指定 `NSKeyValueObservingOptions`，以监听 `BJLRoom` 的 `liveStarted` 属性为例。

```
let observingOptions : NSKeyValueObservingOptions = [.new, .old, .initial]
self.bjl_kvo(BJLPropertyMeta.instance(withTarget: self.room, name: "liveStarted"),
             options: observingOptions,
             observer: {(new:Any?, old:Any?, change:Any?) -> Bool in
                if let liveStarted = new as? Bool {
                    // your code
                }
                return true
             })
```

3.3 protocol

以 `BJLRoom` 的 `slideshowViewController` 属性为例，它在 OC 中定义如下：

```
/** 课件、画笔视图
 尺寸、位置随意设定 */
@property (nonatomic, readonly, nullable) UIViewController<BJLSlideshowUI>
```

```
*slideshowViewController;
```

在 Swift 项目中编译之后变为：

```
/** 课件、画笔视图
    尺寸、位置随意设定 */
 open var slideshowViewController:
UIViewController? { get }
```

可以发现，`slideshowViewController` 作为 `BJLRoom` 的属性时所遵循的 `BJLSlideshowUI` 这个 `protocol` 被 Swift 忽略了，这将导致它在 Swift 中无法调用 `BJLSlideshowUI` 中声明的属性和方法。

解决方法：

以上述 `slideshowViewController` 为例，使用如下方式调用 `BJLSlideshowUI` 中定义的属性和方法：

```
if let slideShowUI:BJLSlideshowUI =
room.slideshowViewController as? BJLSlideshowUI
{
   slideShowUI.contentMode =
BJLContentMode.scaleAspectFill
}
```

附录

API 文档

- API 文档