iOS 直播 Core SDK

git 链接：http://git.baijiashilian.com/open-ios/BJLiveCore.git

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

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, 该 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。

1. 概念

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，不包含上、下课等改变教室状态的功能
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
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
聊天	直播间内的群聊功能，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（需要在 PC 端上传）
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

2. 主要功能

模块	功能	对应文件
教室管理	进入 / 退出教室及相应的事件监听	BJLRoom.h
	断开重连	
	进入教室的加载状态监听	BJLLoadingVM.h
	老师：上课 / 下课	BJLRoomVM.h
在线用户信息管理	加载在线用户信息	BJLOnlineUsersVM.h
	监听用户进入、退出教室	
音视频采集	开启 / 关闭音视频采集	BJLRecordingVM.h
	音视频采集状态监听	
	采集设置：视频方向，清晰度，美颜	
视频播放	播放、关闭指定用户的视频	BJLPlayingVM.h
	老师：远程开关用户麦克风、摄像头	
	监听对象音视频开关状态、音视频用户列表变化	
音视频链路设置	设置用于音视频的 上行 / 下行 链路的类型：UDP / TCP	BJLMediaVM.h
举手、发言邀请	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	

模块	功能	对应文件
课件管理	上传、添加课件，删除课件，课件翻页	BJLSlideVM.h
	加载所有课件	BJLSlideshowVM.h
	监听课件添加、删除	
画笔	开启/关闭画笔，清空画板	BJLSlideshowVM.h
聊天	发送消息（文字、图片、表情）	BJLChatVM.h
	监听收到消息	
	禁言	
录课	老师：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
公告	发布公告	BJLRoomVM.h
	获取教室公告，监听公告变化	
测验	获取历史题目、新题目、答题统计	BJLRoomVM.h
	学生：答题	

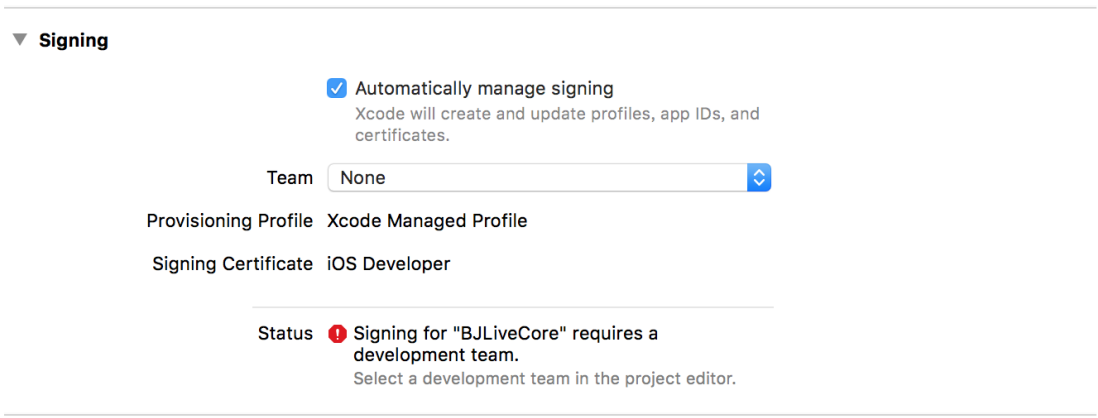
Demo

1. Demo 源文件

在 git 上下载最新版本的 SDK，demo 源文件 在 `BJLiveCore/demo/BJLiveCore` 文件夹中。

2. Demo 编译、运行

- 在 demo 的工程目录下执行 `pod install`
- 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，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。

引入 SDK

SDK 支持 iOS 8.0 及以上 的系统，iPhone、iPad 等设备，集成 1.0.0 或以上版本的 SDK 要求 Xcode 的版本至少为 9.0，它会依赖一些第三方库，建议使用 CocoaPods 方式引入：

- Podfile 中设置 source

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

- Podfile 中引入 BJLiveCore

```
pod 'BJLiveCore', '~> 1.0'

# 用于动态引入 Framework，避免冲突问题
script_phase \
:name => '[BJLiveCore] Embed Frameworks',
:script => 'Pods/BJLiveCore/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/BJLiveBase/script/ClearArchsFromFrameworks.sh "BJHLMediaPlayer.framework"',
:execution_position => :after_compile
```

- 工程目录下执行 `pod install`

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很不稳定，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 'BJLiveCore', '1.0.0'
```

- 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 'BJLiveCore', '~> 1.0.0'
```

- 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 'BJLiveCore', '~> 1.0'
```

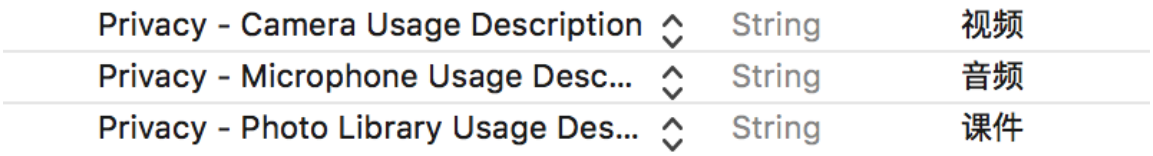
- 激进型(不推荐)：`pod update` 时会升级大版本，可能导致编译出错、必须调整代码，升级后需要 严格测试

```
pod 'BJLiveCore'
```

工程设置

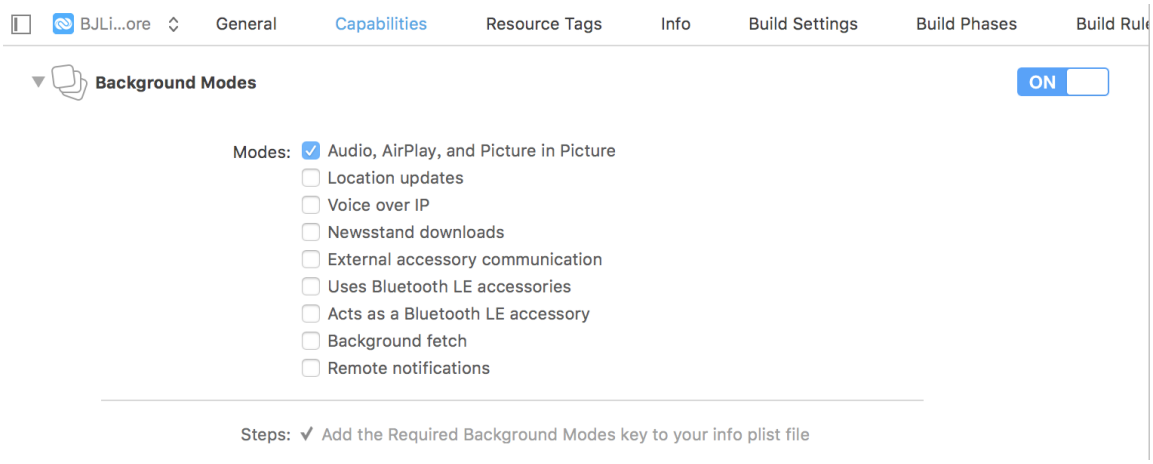
- 隐私权限：在 `Info.plist` 中添加麦克风、摄像头、相册访问描述

```
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
```




```
<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 所有的直播功能都是基于教室这个场景的，进入教室成功之后才能正常使用各个功能模块。要进入教室，需要创建 `BJLRoom` 的一个实例，调用相关进入方法，同时，SDK 也提供进入教室的加载过程的监听方法、支持断点重连。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，可根据需要自行选择。流程要点总结如下：

- `BJLRoom` 是直播功能的入口，用于处理创建、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 都可为空；
- 调用 `enter` 方法后 `room.loadingVM` 将被初始化，可用于显示加载度、成功和失败等，成功/失败后为空；
- 其它 VM 在 loading 过程被初始化，当 `vmsAvailable` 变为 `YES` 时 可开始监听 VM 的属性、方法调用，在进入教室后、`inRoom` 变为 `YES` 时可调用方法、发起请求；
- 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（如有用户退出教室）。

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`：我们没有过多地使用 `method-swizzling`，self 和被监听对象 dealloc 时都会自动取消监听。

1.1.1 Block KVO

- KVO 调用方式，支持 filter - 可选：

```
@weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room.roomVM, // 对象
liveStarted) // 属性名，支持代码自动完成
filter:^BOOL(NSNumber *old, NSNumber *now) { // 过滤
return old.boolValue != now.boolValue; // 返回 NO 丢弃
}
observer:^BOOL(NSNumber *old, NSNumber *now) { // 处理
@strongify(self);
// console 为自定义的控制台视图
[self.console printFormat:@"liveStarted: %@", NSStringFromBOOL(now.boolValue)];
return YES; // 返回 NO 停止监听
}];
```

- 支持两种方式取消某次 KVO

```
@weakify(self);
id<BJLObservation> observation =
[self bjl_kvo:BJLMakeProperty(self.room.roomVM, liveStarted)
observer:^BOOL(NSNumber *old, NSNumber *now) {
@strongify(self);
[self.console printFormat:@"liveStarted: %@", NSStringFromBOOL(now.boolValue)];
return YES; // 1. 返回 NO 取消 KVO
}];
[observation stopObserving]; // 2. 取消 KVO
```

1.1.2 Block 监听方法调用

- 监听方法调用，支持 filter - 可选：

```
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, // 对象
roomWillExitWithError:) // 方法
filter:(BJLMethodFilter)^BOOL(BJLError *error) { // 过滤
return !!error; // 返回 NO 丢弃
}];
observer:(BJLMethodObserver)^BOOL(BJLError *error) { // 处理
@strongify(self);
[self.console printFormat:@"roomWillExitWithError: %@", error];
return YES; // 返回 NO 停止监听
}];
```

- 支持 多个参数：

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

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

```
@weakify(self);
id<BJLObservation> observation =
[self bjl_observe:BJLMakeMethod(self.room, roomWillExitWithError:)
observer:(BJLMethodObserver)^BOOL(BJLError *error) {
@strongify(self);
[self.console printFormat:@"roomWillExitWithError: %@", error];
return YES; // 1. 返回 NO 取消监听
}];
[observation stopObserving]; // 2. 取消监听
```

2. 引入头文件

```
#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

- 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。

```
NSString *domainPrefix = @"yourDomainPrefix";
[BJLRoom setPrivateDomainPrefix:domainPrefix];
```

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
*/
self.room = [BJLRoom roomWithSecret:roomSecret
userName:userName
userAvatar:nil];
```

3.4 准备进入教室：添加状态监听

- 监听进入、退出教室等事件

```objc
// 监听进入教室成功
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, enterRoomSuccess)
observer:^BOOL() {
@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 *old, NSNumber *now) {
return old.boolValue != now.boolValue;
}
observer:^(NSNumber *old, NSNumber *now) {
//上课状态发生变化后的响应操作
NSLog(@"liveStarted: %@", now.boolValue? @"YES" : @"NO");
}];
}
// 处理进教室后的逻辑
[self didEnterRoom];
return YES;
}];
```

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

```objc
// 监听准备退出教室，error 为 nil 表示主动退出
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, roomWillExitWithError:)
observer:^BOOL(BJLError *error) {
@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
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, roomDidExitWithError:)
observer:^BOOL(BJLError *error) {
@strongify(self);
if (error) {
// 获取错误信息
NSString *message = error ? [NSString stringWithFormat:@"%@ - %@",
error.localizedDescription,
error.localizedFailureReason] : @"错误";
NSLog(@"error: %@", message);
}
[self.room exit];
return YES;
}];
```

- 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
@weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room, loadingVM)
filter:^BOOL(id old, id now)
return !!now;
}
observer:^BOOL(id old, BJLLoadingVM *now) {
@strongify(self);
// 自定义方法，处理当前的加载任务，可参考 demo
[self makeEventsForLoadingVM:now];
return YES;
}];
```

```
// 断开重连
@weakify(self);
[self.room setReloadingBlock:^(BJLLoadingVM * _Nonnull reloadingVM, void (^ _Nonnull callback)
(BOOL)) {
@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);
}];
}];
```

```
// 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
*/
@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)) {
@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"
```

```
46. style:UIAlertActionStyleDefault
47. handler:^(UIAlertAction * _Nonnull action) {
48. continueCallback(YES);
49. }]];
50. [alert addAction:[UIAlertAction
51. actionWithTitle:@"取消"
52. style:UIAlertActionStyleCancel
53. handler:^(UIAlertAction * _Nonnull action) {
54. [self exitRoom];
55. continueCallback(NO);
56. }]];
57. [self presentViewController:alert animated:YES completion:nil];
58. }
59. };
```

3.5 进出教室

```
1. // 进入教室
2. [self.room enter];
3.
4. // 退出教室
5. [self.room exit];
```

3.6 上下课

```
1. // 上课
2. BJLError *error = [self.room.roomVM sendLiveStarted:YES];
3.
4. // 下课
5. BJLError *error = [self.room.roomVM sendLiveStarted:NO];
```

3.7 教室内在线用户信息获取（仅限 100 人以内）

在线用户信息列表采用分页加载，参考 BJLOnlineUsersVM

- 监听在线用户变化

```
1. [self bjl_kvo:BJLMakeProperty(self.room.onlineUsersVM, onlineUsers)
2. observer:^BOOL(id _Nullable old, id _Nullable now) {
3. bjl_strongify(self);
4. [self updateTitleWithOnlineUsersTotalCount];
5. [self.tableView reloadData];
6. return YES;
7. }];
```

- 监听用户进入教室

```
1. [self bjl_observe:BJLMakeMethod(self.room.onlineUsersVM, onlineUserDidEnter:)
2. observer:^BOOL(BJLUser *user) {
3. @strongify(self);
4. NSlog(@"onlineUsers in: %@", user.name);
5. return YES;
6. }];
```

- 监听用户退出教室

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

3.8 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];
```

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，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 只有老师和助教才能发送定制广播信令
#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_invalidUserRole 当前用户不是老师或者助教
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 60 条
*/
BJLError *error = [self.context.room.roomVM sendCustomizedBroadcast:customizedKey value:value cache:isNeedCache];
```

4. 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

4.1 音视频状态监听

4.1.1 监听采集状态：通过监听 `self.room.recordingVM` 的属性变化及方法调用来实现

- 关键属性监听

```
@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
@property (nonatomic, readonly) CGFloat inputVideoAspectRatio;// 视频采集宽高比
```

```
// 以 inputVolumeLevel 为例
[self bjl_kvo:BJLMakeProperty(self.room.recordingVM, inputVolumeLevel)
options:NSKeyValueObservingOptionOld | NSKeyValueObservingOptionNew
filter:^BOOL(NSNumber * _Nullable old, NSNumber * _Nullable now) {
// 音量变化超过一定程度才触发
return ABS(round(old.doubleValue * 10) - round(now.doubleValue * 10)) >= 1.0;
}
observer:^BOOL(NSNumber * _Nullable old, NSNumber * _Nullable now) {
NSLog(@"current input volume level:%f", now.doubleValue);
return YES;
}];
```

4.1.2 监听播放状态：通过监听 `self.room.playingVM` 的属性变化及方法调用来实现

- 关键属性监听

```
@property (nonatomic, readonly, nullable, copy) NSArray<BJLMediaUser *> *playingUsers;// 音视频用户列表
@property (nonatomic, readonly, nullable) BJLOnlineUser *videoPlayingUser;// 当前播放对象
```

```
bjl_weakify(self);
// 以 videoPlayingUsers 为例，videoPlayingUsers 表示当前正在播放的对象，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 *> *old, NSArray<BJLUser *> *now) {
NSLog(@"You are playing videos of %td users", now.count);
return YES;
}];
// 视频宽高比变化监听
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingViewAspectRatioChanged:forUserWithID:)
observer:(BJLMethodObserver)^BOOL(CGFloat videoAspectRatio, NSString *userID) {
bjl_strongify(self);
self.aspectRatio = videoAspectRatio;
return YES;
}];
```

- 关键方法监听

```
/** 用户开关音、视频 */
- (BJLObservable)playingUserDidUpdate:(BJLOnlineUser *)now
old:(BJLOnlineUser *)old;
```

4.1.3 监听音视频用户列表

`BJLPlayingVM` 的属性 `playingUsers` 表示当前正在使用音频、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：

```
[self bjl_kvo:BJLMakeProperty(self.room.playingVM, playingUsers)
observer:^BOOL(id _Nullable old, id _Nullable now) {
NSLog(@"playing users changed");
return YES;
}];
```

4.2 音视频采集

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。

4.2.1 采集控制

- 基本操作

```
/** 开关音视频
#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
#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
#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`
发言人数已达上限，参考 `room.featureConfig.maxSpeakerCount`、`room.playingVM.playingUsers.count`
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];
```

- 采集音视频

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

// 采集音频、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）
BJLError *error = [self.room.recordingVM setRecordingAudio:YES recordingVideo:YES];
```

- 如果上麦路数达到上限，会导致开启采集音视频被拒绝

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

[self bjl_observe:BJLMakeMethod(self.room.recordingVM, remoteChangeRecordingDidDenyForUser:)
observer:^BOOL(BJLUser *user) {
bjl_strongify(self);
[self showProgressHUDWithText:[NSString stringWithFormat:@"服务器拒绝强制 %@ 发言，音视频并发已达上限", user.name]];
return YES;
}];
```

- 设置全体禁止采集音频

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

- 停止采集

```
// 关闭音视频采集
BJLError *error = [self.room.recordingVM setRecordingAudio:NO recordingVideo:NO];

// UI：移除 recordingView，注意不要释放
[self.room.recordingView removeFromSuperView];
```

4.2.2 采集设置

- 禁止采集声音：`BJLRecordingVM.h` 中定义了相关属性 `forbidRecordingAudio`，`forbidAllRecordingAudio`

```
/** 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;
```

- 切换摄像头：`BJLRecordingVM.h` 中定义了相关属性 `usingRearCamera`

```
/** 是否使用后置摄像头 */
@property (nonatomic) BOOL usingRearCamera; // NO: Front, YES Rear(iSight)
```

```
// 切换摄像头：直接改变 usingRearCamera 的值即可，SDK 内部会作出相应处理
self.room.recordingVM.usingRearCamera = !self.room.recordingVM.usingRearCamera;
```

- 清晰度设置：`BJLRecordingVM.h` 中定义了相关属性 `videoDefinition`

```
/** 清晰度 */
@property (nonatomic) BJLVideoDefinition videoDefinition;
```

```
/** 设置清晰度为高清
BJLVideoDefinition_low 流畅
BJLVideoDefinition_high 高清
BJLVideoDefinition_default 默认
* /
self.room.recordingVM.videoDefinition = BJLVideoDefinition_high;
```

- 美颜设置：`BJLRecordingVM.h` 中定义了相关属性 `videoBeautifyLevel`

```
/** 美颜 */
@property (nonatomic) BJLVideoBeautifyLevel videoBeautifyLevel;
```

```
/** 设置美颜等级
BJLVideoBeautifyLevel_0,
BJLVideoBeautifyLevel_1,
.....
BJLVideoBeautifyLevel_off
*/
self.room.recordingVM.videoBeautifyLevel = BJLVideoBeautifyLevel_on;
```

4.3 音视频播放

学生只能选择自己是否播放用户的视频，发言用户的音频默认播放，无法控制；老师则有权限远程开关用户的麦克风、摄像头，这将影响到所有用户的播放。播放视频时，可通过监听音视频用户列表的变化，即时获取最新列表，便于播放对应用户视频，参考监听音视频用户列表。

- 基本操作

```
// 播放 / 关闭 单个用户的视频：调用 BJLPlayingVM 的 updatePlayingUserWithID:videoOn:方法，参数 videoOn 为 YES 表示打开视频，为 NO 则表示关闭视频。user 从 BJLPlayingVM 的 playingUsers 获取（playingUsers 随时可能变化，需要监听它的变化、在变化回调中取值，参考“监听音视频变化列表”部分）
BJLError * error = [self.room.playingVM updatePlayingUserWithID:user.ID videoOn:YES]; // YES：播放，NO：关闭
```

- 自动播放视频

```
/** 自动播放视频回调
#discussion 其他用户视频可用时调用，返回 YES 表示自动播放视频，不设置此 block 不会自动播放
*/
@property (nonatomic, copy, nullable) BOOL (^videoPlayingBlock)(BJLMediaUser *user);
```

- 播放视频：将用户的 ID 作为参数，播放指定用户的视频。这里以学生播放老师视频为例（老师身份的 `user` 可以通过 `self.room.onlineUserVM.onlineTeacher` 更快速的获取，这里只是演示从 `playingUsers` 中选择一个用户的视频进行播放）：

```
// 从音视频用户列表 playingUsers 中筛选出老师
for (BJLOnlineUser *user in self.room.playingVM.playingUsers) {
if (user.isTeacher && user.videoOn) { // 身份为老师且开启了视频
// 获取对应用户的播放视图
UIView *playingView = [self.room.playingVM playingViewForUserWithID:user.ID];
// 将播放视图添加到当前 viewController 的对应视图（布局自定）
[self.playingView addSubview:playingView];
// 播放视频
[self.room.playingVM updatePlayingUserWithID:user.ID videoOn:YES]
break;
}
}
```

- 播放媒体，共享桌面

```
/** 老师在 PC 上更改共享桌面设置、媒体文件播放状态
#discussion 这两个属性需要与老师的在线状态、音视频状态配合使用
*/
@property (nonatomic, readonly) BOOL teacherSharingDesktop, teacherPlayingMedia;
```

- 多个清晰度播放选择

```
@property (nonatomic, copy, nullable)
BJLTupleType(BOOL autoPlay, NSInteger definitionIndex)
(^autoPlayVideoBlock)(BJLMediaUser *user, NSInteger cachedDefinitionIndex);

/** 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;

/** 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;
```

- 停止播放

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

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

- 可以监听初始化播放, 播放成功，如果用户开启了视频，可以在这期间显示用户头像或者加载动画来过渡视频渲染的时间

```
/** 将要播放音视频
#discussion 播放或者关闭音视频的方法被成功调用
#param playingUser 将要播放音视频用户
*/
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidStartLoadingVideo:)
observer:^BOOL(BJLMediaUser *user) {
bjl_strongify(self);
[self tryToShowLoadingViewWithUser:user];
return YES;
}];
/** 播放成功
#discussion 用户音视频开启或者关闭成功
#param playingUser 播放音视频的用户
*/
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playingUserDidFinishLoadingVideo:)
observer:^BOOL(BJLMediaUser *user) {
bjl_strongify(self);
[self tryToCloseLoadingViewWithUser:user];
return YES;
}];
```

- 视频播放出现卡顿

```
/** 播放出现卡顿
@param userID 出现卡顿的正在播放的视频用户ID
*/
//bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.playingVM, playLagWithPlayingUserID:)
observer:^BOOL{
//bjl_strongify(self);
NSLog(@"当前网络状况较差");
return YES;
}];
```

4.4 音视频设置

- 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。
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

```
/**
设置上、下行链路 (SDK 内部将监听属性值的变化，进行相应的切换处理)
BJLLinkType_TCP：TCP
BJLLinkType_UDP：UDP
*/
// 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];
```

- 音视频采集，播放控制，SDK对前后台切换进行了处理，只要修改这些配置项即可

```
/** 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) BOOL supportBackgroundAudio;
/** 是否支持后台采集声音, 默认不支持 */
@property (nonatomic) BOOL supportBackgroundRecordingAudio;
/** 是否播放视频静音, 默认不静音 */
@property (nonatomic) BOOL needMutePlayingAudio;
```

5. 举手发言

5.1 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` 方法获取）。

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

```
@property (nonatomic, readonly) BOOL forbidSpeakingRequest;
```

- 判断当前是否是发言状态

```
/** 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，要求学生权限。
*/
BJLError *error = [self.room.speakingRequestVM sendSpeakingRequest];
```

- 监听举手发言申请的处理结果

```
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestVM,
speakingRequestDidReplyEnabled:isUserCancelled:user:)
observer:(BJLMethodObserver)^BOOL(BOOL speakingEnabled, BOOL isUserCancelled, BJLUser *user)
{
@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;
}];
/** 学生: 举手发言申请被自动拒绝，因为上麦路数达到上限 */
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestVM, speakingRequestDidDeny)
observer:^BOOL(void) {
@strongify(self);
[self showProgressHUDWithText:@"服务器拒绝申请发言，音视频并发已达上限"];
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:recordingVideo:recordingAudioChanged:recordingVideoChanged:`
#param enabled YES：开启，NO：关闭
*/
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestVM, 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

```
/** 超时时间及更新频率：定义在 SDK 内部，外部可访问。
调用 sendSpeakingRequest 方法举手时设置超时时间为 BJLSpeakingRequestTimeoutInterval （默认为30，可通过后台配置）秒，
每 BJLSpeakingRequestCountdownStep （固定为0.1）秒更新，
变为 0.0 时自动取消举手
*/
extern const NSTimeInterval BJLSpeakingRequestTimeoutInterval, BJLSpeakingRequestCountdownStep;
```

```
// 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.speakingRequestVM, speakingRequestTimeRemaining)
options:NSKeyValueObservingOptionOld | NSKeyValueObservingOptionNew
filter:^BOOL(NSNumber * _Nullable old, NSNumber * _Nullable timeRemaining) {
return timeRemaining.doubleValue != old.doubleValue;
}
observer:^BOOL(NSNumber * _Nullable old, NSNumber * _Nullable timeRemaining) {
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];
```

5.2 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.speakingRequestVM, didReceiveSpeakingInvite:)
observer:^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
```

5.3 老师处理发言申请

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

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

- 允许 / 拒绝发言

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

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

```
@weakify(self);
[self bjl_observe:BJLMakeMethod(self.room.speakingRequestVM, receivedSpeakingRequestFromUser:)
observer:^BOOL(BJLUser *user) {
@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
#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`
#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
#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];

```

6. 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，但目前只支持图片格式的文件上传，其他格式的文档需要通过调用后台 API 或者使用 PC客户端上传添加。

- 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。

```
// 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
```

- 设置课件尺寸

```
// 默认720，范围 (1 ~ 4096)
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.slideshowViewController];
[self.slideshowView addSubview:self.room.slideshowViewController.view];
[self.room.slideshowViewController didMoveToParentViewController:self];
```

```
/** 设置显示模式
BJLContentMode_scaleAspectFit 完整
BJLContentMode_scaleAspectFill 铺满
BJLContentMode_scaleToFill 拉伸
*/
self.room.slideshowViewController.contentMode = BJLContentMode_scaleAspectFit;
```

- 上传、添加课件：目前只支持图片格式，`fileURL` 为图片的文件路径

```
@weakify(self);
[self.room.slideVM uploadImageFile:fileURL
progress:^(CGFloat progress){
@strongify(self);
// 显示进度
self.progressView.progress = progress;
}
finish:^(BJLDocument * _Nullable document, BJLError * _Nullable error) {
@strongify(self)
if(document){
[self.room.slideVM addDocument:document];
}
else{
NSLog(@"error:%@", error);
}
}];
```

- 删除课件

```
// 根据 ID 删除课件
[self.room.slideVM deleteDocunmentWithID:documentID];
```

- 监听课件变化：通过监听 `self.room.slideshowVM` 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
@weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room.slideshowVM, allDocuments)
observer:^BOOL(id _Nullable old, NSArray<BJLDocument *> * _Nullable now) {
@strongify(self);
// 更新数据源及相关界面控件
self.allDocuments = [now mutableCopy];
[self.tableView reloadData];
[self updateViewsForDataCount];
return YES;
}];
```

```
// 以监听添加课件的通知为例
[self bjl_observe:BJLMakeMethod(self.room.slideshowVM, 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;
}];
```

```
// 以监听课件教室内当前页变化为例：
[self bjl_kvo:BJLMakeProperty(self.room.slideshowVM, currentSlidePage)
observer:^BOOL(BJLSlidePage * _Nullable old, BJLSlidePage * _Nullable now) {
NSLog(@"currentPage：%td", now.documentPageIndex);
return YES;
}];
```

- 监听本地课件页码：`BJLSlideshowVM` 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。本地课件页码通过监听 `BJLRoom` 的 `slideshowViewController` 的 `localPageIndex` 获得。

```
[self bjl_kvo:BJLMakeProperty(self.room.slideshowViewController, localPageIndex)
observer:^BOOL(id _Nullable old, id _Nullable now) {
NSLog(@"localPage：%td", [now integerValue]);
return YES;
}];
```

7. 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，操作画笔时用户的当前课件页面必须与老师保持一致。

- 显示画笔视图: 目前画笔与课件共用同一个视图 `self.room.slideshowViewController.view`
- 开启、关闭画笔

```
self.room.slideshowViewController.drawingEnabled = YES; // YES:开启，NO:关闭
```

- 画笔授权

```
/** 开启、关闭画笔
开启画笔时，如果本地页数与服务端页数不同步则无法绘制
`drawingGranted` 是 YES 时才可以开启，`drawingGranted` 是 NO 时会被自动关闭
#param drawingEnabled YES：开启，NO：关闭
#return BJLError:
BJLErrorCode_invalidCalling 错误调用，当前用户是学生、`drawingGranted` 是 NO
*/
- (nullable BJLError *)updateDrawingEnabled:(BOOL)drawingEnabled;
- (void)setDrawingEnabled:(BOOL)drawingEnabled DEPRECATED_MSG_ATTRIBUTE("use `updateDrawingEnabled:` instead");
```

- 清空画板

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

8. 聊天

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

- 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。
- 获取所有消息

```
[self.allMessages removeAllObjects];
[self.allMessages addObjectsFromArray:self.room.chatVM.receivedMessages];
```

- 监听收到消息的通知

```
[self bjl_observe:BJLMakeMethod(self.room.chatVM, didReceiveMessage:)
observer:^BOOL(BJLMessage *message) {
NSLog(@"%@: %@", message.fromUser.name, message.content);
return YES;
}
```

- 禁言

```
// 老师设置全体禁言状态
[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;
```

```
// 老师禁言单个用户，可设置禁言时长
[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;
}];
```

- 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
@weakify(self);
// 上传图片
[self.room.chatVM uploadImageFile:fileURL
progress:^(CGFloat progress){
@strongify(self);
// 显示进度
self.imageUploadingView.progress = progress;
}
finish:^(NSString * _Nullable imageURLString, BJLError * _Nullable error) {
@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]];
}
}
```

9. 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 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

```
BOOL serverRecording = self.room.serverRecordingVM.serverRecording;
```

- 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
```

- 监听云端录课不可用的通知

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

10. 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

```
@weakify(self);
[self bjl_kvo:BJLMakeProperty(self.room.roomVM, notice)
observer:^BOOL(id _Nullable old, BJLNotice * _Nullable notice) {
@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];
```

11. 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。

- 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;
}];
```

- 学生：收到新题目

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

- 学生答题：支持多选。

```
/**
@param answers 学生选择的 BJLSurveyOption 的 key 的数组，
@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
@param order 序号，BJLSurvey 的 order
*/
[self.room.roomVM sendSurveyAnswers:answers result:result order:order];
```

```
// 例：假设当前题目为 survey，学生选择的 BJLSurveyOption 的 key 的数组为 selectAnswers
NSMutableArray *answers;
BJLSurveyResult result;
// 筛选出当前题目的所有正确答案
for (BJLSurveyOption *option in survey.options) {
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. 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，下课清空点赞记录

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

- 点赞用户

```
/**
点赞
#discussion 点赞时需要把传历史所有点赞记录, 包括本次点赞
#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;
}];
```

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，对于 `viewModel` 类型（如 `roomVM`、`playVM` 等）的属性，可在监听到 `BJLRoom` 的 `vmsAvailable` 属性为 `YES` 之后添加对它们的相关监听；其他类型的属性，大部是在进入教室成功之后实例化的，在此之前添加的监听无效，可在监听到 `BJLRoom` 的 `enterRoomSuccess` 方法被调用后添加相关监听。
- 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:

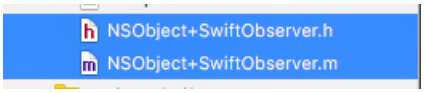
- `playingUsers` 是随时变化的，不能用直接取值的方法来获取音视频列表，应该监听 `self.room.playingVM` 的 `playingUsers` 属性的变化，即时获取最新列表，参考监听音视频用户列表。使用监听方式出现此问题则请参考后续部分。
- 检查添加监听时 `BJLRoom` 的 `vmsAvailable` 属性是否为 `YES`
- 检查添加监听时 `BJLPlayingVM` 的实例 `self.room.playingVM` 是否为空
- 检查教室内是否有用户在发言（打开了音频或视频）

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 = [.old, .new, .initial]
self.bjl_kvo(BJLPropertyMeta.instance(withTarget: self.room, name: "liveStarted"),
options: observingOptions,
observer: {(old:Any?, new: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` 中定义的属性和方法。

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
}
```

4. 集成后发现SDK存在冲突（ffmpeg版本冲突）

- 解决方法:
  - 第1步: 修改podfile里面的依赖
    - 修改或者增加集成的点播base仓库 `pod 'BJLiveCore', '(当前集成的正式版的版本号)-weak'`
  - 第2步, 修改podfile里面的依赖之后执行 `pod update`
  - 第3步, 获取百家云sdk的 `BJHLMediaPlayer.framework`, 步骤:
    - 从 http://git.baijiashilian.com/open-ios/BJLiveCore 下载对应版本版本的代码
    - 取出 BJHLMediaPlayer.framework, 然后复制到本地工程的文件夹里面
  - 第4步, 将BJHLMediaPlayer.framework手动拖入工程, 然后:
    - 进入项目的target -> General -> Linked Frameworks and Libraries, 删除 `Linked Frameworks and Libraries` 下的 `BJHLMediaPlayer.framework`.
    - 进入项目的target -> General -> Embedded Binaries (*注意: 不是* `Embedded Frameworks`), 添加 `BJHLMediaPlayer.framework`, 步骤:
      - 点击 `Embedded Binaries` 下的 "+" 号
      - 点击 "Add Other..."
      - 找到本地`BJHLMediaPlayer.framework`, 添加即可

附录

1. 变更记录

- ChangeLog

2. API 文档

- API 文档