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。
- 在自定义的viewController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，比如参加吗或者签名的方式。
- 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。

教室管理
进入教室的加载状态监听
进入 / 退出教室及相应的事件监听
断开重连
大小班切换
老师或助教：上课 / 下课

定制信令

1. 设置专属域名前缀

- BJLiveCore SDK 1. 5 及之后版本支持设置专属域名前缀。
- 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.baijiayun.com，则前缀为 demo123，更多细节参考 专属域名说明。建议在每次创建直播间之前确保专属域名设置正确

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

2. 定义教室属性

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

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 number      用户编号，合作方账号体系下的用户 ID 号，必须是数字
#param name        用户名
#param groupID     分组 ID，不分组参与了签名计算时传 0，未参与签名计算时传 NSNotFound
#param avatar      用户头像 URL，可传空值
#param role        用户角色：老师、学生等
```

```
#param roomID       教室 ID
#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];
```

4. 加载教室之前

Core SDK 提供了一些属性在进教室之前可以预先设置，此处只是简单举例，具体更详细的设置可以查看不同viewModel提供的更多属性。

```objc
- (void)prepareForEnterRoom {
// 采集设置

self.room.recordingVM.videoRecordingOrientation = BJLVideoRecordingOrientation_alwaysPortrait;
    self.room.recordingVM.videoContentMode = BJLVideoContentMode_aspectFill;

// 播放设置
    self.room.playingVM.videoContentMode = BJLVideoContentMode_aspectFill;
    bjl_weakify(self);
    self.room.playingVM.autoPlayVideoBlock = ^BJLAutoPlayVideo(BJLMediaUser *user, NSInteger cachedDefinitionIndex) {
        bjl_strongify(self);
        // 根据内部逻辑更新user是否需要自动播放视频
        NSString *videoKey = [self videoKeyForUser:user];
        BOOL autoPlay = videoKey && ![self.autoPlayVideoBlacklist containsObject:videoKey];
        NSInteger definitionIndex = cachedDefinitionIndex;
        if (autoPlay) {
            NSInteger maxDefinitionIndex = MAX(0, (NSInteger)user.definitions.count - 1);
            definitionIndex = (cachedDefinitionIndex <= maxDefinitionIndex
                               ? cachedDefinitionIndex
                               : maxDefinitionIndex);
        }
        return BJLAutoPlayVideoMake(autoPlay, definitionIndex);
    };
```

```
23.
24. // 自定义大班课白板，如不设置，则会使用默认的白板数据
25.     BJLWhiteboard *wb = [BJLWhiteboard new];
26.     wb.width = xxx;
27.     wb.height = xxx;
28.     wb.urlString = xxx;
29.     wb.name= xxx;
30.     self.room.documentVM.whiteboard = wb;
31.
32. // 小班课黑板和小黑板自定义背景
33.     self.room.documentVM.blackboardImage = [UIImage
    bjl_imageWithColor:BJLTheme.blackboardColor];
34.     self.room.documentVM.writingBoardImage = [UIImage
    bjl_imageWithColor:BJLTheme.windowBackgroundC
35.
36. }
```

5. 进/出教室成功/失败监听

- 监听进入、退出教室等事件。

```
1. // makeObservingBeforeEnterRoom
2.
3. bjl_weakify(self);
4.
5. // 监听到教室信息加载
6. [self bjl_kvo:BJLMakeProperty(self.room, roomInfo)
7.         observer:^BJLControlObserving(id _Nullable value, id _Nullable oldValue, BJLPropertyChange *_Nullable change) {
8.             bjl_strongify(self);
```

```
            if (self.room.roomInfo) {
                // 教室信息获取成功
            }
            return YES;
        }];
// 监听进入教室成功
[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];
    }

    // 进入教室成功需要设置断网重连block
    [self.room setReloadingBlock:xxx];

    // 处理进教室后的逻辑
    [self didEnterRoom];
    return YES;
}];

[self bjl_kvo:BJLMakeProperty(self.room, reloading)
        filter:^BJLControlObserving(id _Nullable value, id _Nullable oldValue, BJLPropertyChange *_Nullable change) {
            return [value boolValue] != [oldValue boolValue];
        }
        observer:^BJLControlObserving(id _Nullable value, id _Nullable oldValue, BJLPropertyChange *_Nullable change) {
            bjl_strongify(self);
            if (self.room.reloading) {
```

```
            [self showProgressHUDWithText:BJLLocalizedString(@"即将重新进入直播间")];
        }
        else {
            [self showProgressHUDWithText:BJLLocalizedString(@"重新连接成功")];
        }
        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;
}];
```

6. 断线重连

```
/**
 断开、重连
 #discussion 网络连接断开时回调，回调 callback 确认是否重连，YES 重连、NO 退出直播间，也可延时或者手动调用 callback
 #discussion 可通过 `reloadingVM` 监听重连的进度和结果
 #discussion 默认（不设置此回调）在断开时自动重连、重连过程中遇到错误将 `异常退出`
 #discussion !!!: 断开重连过程中 vm 的状态、数据没有与服务端同步，调用其它 vm 方法时发起的网络请求会被丢弃、甚至产生不可预期的错误
```

```
 #param reloadingBlock 重连回调。reloadingVM：重连 vm；callback(reload)：调用 callback 时 reload 参数传 YES 重连，NO 将导致 `异常退出`
 */
- (void)setReloadingBlock:(void (^_Nullable)(BJLLoadingVM *reloadingVM,
                                             void (^callback)(BOOL reload)))reloadingBlock;
```

连接失败错误信息具体参考 `NSError+BJLError.h` 文件：

```
/**
 - stepOver: 单步完成，无错误
 - askForWWANNetwork: 蜂窝网络，无错误
 - errorOccurred: 发生错误，参考 BJLErrorCode
 */
typedef NS_ENUM(NSInteger, BJLLoadingSuspendReason) {
    /** 单步完成，无错误 */
    BJLLoadingSuspendReason_stepOver,
    /** 发生错误 */
    BJLLoadingSuspendReason_errorOccurred
};
```

直播间加载成功之后，SDK内部会设置 `self.loadingVM = nil` ，然后回调通知 `enterRoomSuccess` 。对于直播中的断网重连场景，需要在收到 `enterRoomSuccess` 回调之后再次设置 `room.reloadingBlock` ，及时响应 `room.reloadingVM` 处理断网的逻辑。

```
// 进入直播间成功再次设置断网重连的 block，所以该回调需要在收到`enterRoomSuccess`回调之后设置
bjl_weakify(self);
[self bjl_observe:BJLMakeMethod(self.room, enterRoomSuccess)
```

```
            observer:^BOOL {
        bjl_strongify(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 ...");
        // 此处需要设置room.reloadingVM 的
suspendBlock，以及room.reloadingVM的加载成功和
失败的回调监听，或者进度条更新回调
                [self
makeObservingForLoadingVM:reloadingVM];
                NSLog(@"网络连接断开：重连");
                callback(YES);
            }
                       cancelCallback:^{
                callback(NO);
            }];
        }];
    }];
```

断网重连示例代码：

```
- (void)makeObservingForLoadingVM:
(BJLLoadingVM *)loadingVM {
    bjl_weakify(self);
    loadingVM.suspendBlock = ^(BJLLoadingStep
step,
        BJLLoadingSuspendReason reason,
        BJLError *error,
        void (^continueCallback)(BOOL isContinue))
{
```

```
        bjl_strongify(self);
        // 成功
        if (reason != BJLLoadingSuspendReason_errorOccurred) {
            continueCallback(YES);
             return;
        }

        // 直接退出
         // 试听结束
        if (error.code == BJLErrorCode_enterRoom_auditionTimeout) {
            continueCallback(NO);
            return;
        }

        // 出错

        if (!error) {
            error = BJLErrorMake(BJLErrorCode_unknown, nil);
        }

        if (error.code == BJLErrorCode_enterRoom_timeExpire) {
            BJLPopoverViewController *popoverViewController = [[BJLPopoverViewController alloc] initWithPopoverViewType:BJLExitViewTimeOut message:[NSString stringWithFormat:BJLLocalizedString(@"直播间已过期")]];
            [self bjl_addChildViewController:popoverViewController superview:self.popoversLayer];
```

```
30.         [popoverViewController.view
bjl_makeConstraints:^(BJLConstraintMaker
*_Nonnull make) {
31.
make.edges.equalTo(self.popoversLayer);
32.         }];
33.         [popoverViewController
setConfirmCallback:^{
34.             bjl_strongify(self);
35.             continueCallback(NO);
36.             [self exit];
37.         }];
38.     }
39.     else {
40.         BJLPopoverViewController
*popoverViewController =
[[BJLPopoverViewController alloc]
initWithPopoverViewType:BJLExitViewConnectFail
message:[NSString
stringWithFormat:BJLLocalizedString(@"网络连接失
败（%td-%td），您可以退出或继续连接"), step,
reason]
detailMessage:self.room.roomInfo.customerSupport
41.         [self
bjl_addChildViewController:popoverViewController
 superview:self.popoversLayer];
42.         [popoverViewController.view
bjl_makeConstraints:^(BJLConstraintMaker
*_Nonnull make) {
43.
make.edges.equalTo(self.popoversLayer);
44.         }];
45.         [popoverViewController
setCancelCallback:^{
46.             bjl_strongify(self);
47.             continueCallback(NO);
```

```
            [self exit];
        }];
        [popoverViewController setConfirmCallback:^{
            continueCallback(YES);
        }];

        if (self.room.recordingVM.screenSharing) {
            [self screenShareMaskVCEventHandler];
            self.needRestartScreenShare = YES;
        }
    }
};

[self bjl_observe:BJLMakeMethod(loadingVM, loadingFailureWithError:)
        observer:^BOOL(BJLError *error) {
    bjl_strongify(self);
    [self roomDidExitWithError:error];
    return YES;
}];

[self bjl_observe:BJLMakeMethod(loadingVM, loadingSuccess)
        observer:^BOOL(void) {
    bjl_strongify(self);
    // 增加重连成功之类的文字提示
    return YES;
}];
}
```

7. 准备进入教室：添加状态监听

直播间的加载是一个耗时的长链接连接的过程，`BJLRoom` 提供了 `BJLLoadingVM` 来处理首次进入直播间整个连接过程，同时提供了 `reloadingVM` 处理直播过程中断网重连的问题。

```
/**
 加载任务回调，在 suspendBlock 中调用 callback(BOOL isContinue) 决定是否继续
 #param step        步骤
 #param reason      暂停原因
 #param error       具体错误
 #param callback    callback(BOOL isContinue)回调
 isContinue: reason 为 `BJLLoadingSuspendReason_errorOccurred` 时 isContinue 表示是否重试当前步骤，否则表示是否执行下一步骤
 suspendBlock 为 nil 时使用 isContinue = (reason != BJLLoadingSuspendReason_errorOccurred)
 */
@property (nonatomic, copy, nullable) BJLLoadingSuspendBlock suspendBlock;

/**
 加载进度
 #param progress 进度
 */
- (BJLObservable)loadingUpdateProgress:(CGFloat)progress; // progress: 0.0 ~ 1.0

/** 加载成功 */
- (BJLObservable)loadingSuccess;

/**
 加载失败
 参考 `BJLErrorCode`
```

```
 #param error 错误信息
 */
- (BJLObservable)loadingFailureWithError:(nullable BJLError *)error;
```

一般建议UI上定义一个专门的加载视图，参考UI SDK 的 BJLLoadingViewController的实现以及使用场景。

以 `BJLLoadingViewController` 实现为参考，首先设置进教室前的监听：

```
// 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);
    if (self.room.loadingVM) {
        // 此处需要设置self.room.loadingVM 的suspendBlock，以及self.room.loadingVM的加载成功和失败的回调监听，或者进度条更新回调
                [self makeObservingForLoadingVM:self.room.loadingVM isReload:NO];
    }
    return YES;
}];

// 首次进教室途中可能会加载失败，此处需要
```

```
    [self.room setReloadingBlock:^(BJLLoadingVM *reloadingVM, void (^callback)(BOOL reload)) {
        bjl_strongify(self);
        // 此处需要设置room.reloadingVM 的suspendBlock，以及room.reloadingVM的加载成功和失败的回调监听，或者进度条更新回调
            [self makeObservingForLoadingVM:reloadingVM isReload:YES];
        callback(YES);
    }];

// 首次进教室失败可以在当前页面上更新错误信息
[self bjl_observe:BJLMakeMethod(self.room, enterRoomFailureWithError:)
            observer:^BOOL(BJLError *error) {
                bjl_strongify(self);
                [self makeLoadFailedView];
                return YES;
            }];
```

```
- (void)makeObservingForLoadingVM:(nullable BJLLoadingVM *)loadingVM isReload:(BOOL)reload {
    bjl_weakify(self);
    loadingVM.suspendBlock = ^(BJLLoadingStep step,
        BJLLoadingSuspendReason reason,
        BJLError *error,
        void (^continueCallback)(BOOL isContinue)) {
        // 成功
        if (reason != BJLLoadingSuspendReason_errorOccurred) {
            continueCallback(YES);
```

```
            return;
        }

        // 直接退出
        if (error.code == BJLErrorCode_enterRoom_auditionTimeout) {
            continueCallback(NO);
            return;
        }
        self.error = error;
        [self makeLoadFailedView];
        self.reloadingCallback = ^{
            bjl_strongify(self);
            self.reloadingCallback = nil;
            continueCallback(YES);
        };
    };

    [self bjl_observe:BJLMakeMethod(loadingVM, loadingUpdateProgress:)
             observer:(BJLMethodObserver) ^BOOL(CGFloat progress) {
                 bjl_strongify(self);
                 self.enterRoomProgressLabel.textColor = BJLTheme.brandColor;
                 self.enterRoomProgressLabel.text = [NSString stringWithFormat:BJLLocalizedString(@"正在进入直播间 %.0f%%"), (progress / 1.0) * 100];
                 return YES;
             }];

    /** 首次加载成功进入直播间之后，self.loadRoomInfoSucessCallback回调通知创建UI，隐藏 loading
     */
```

```
    [self bjl_observe:BJLMakeMethod(loadingVM, loadingSuccess)
            observer:^BOOL {
                bjl_strongify(self);
                if (!reload && self.loadRoomInfoSucessCallback) {
                    self.loadRoomInfoSucessCallback();
                }
                [self hide];
                return YES;
            }];

    [self bjl_observe:BJLMakeMethod(loadingVM, loadingFailureWithError:)
            observer:^BOOL(BJLError *error) {
                bjl_strongify(self);
                self.error = error;
                [self makeLoadFailedView];
                return YES;
            }];
}
```

8. 进入 / 退出直播间

在loadingUI准备好之后 可以调用 `enter` 进入直播间，开始链接，加载直播间。

- 进入、退出教室。

```
// 进入教室
[self.room enter];
/** 进入直播间
 #param validateConflict 传 YES 检查是否有相同用户在直播间、如果有则回调错误
```

```
`BJLErrorCode_enterRoom_loginConflict`，传 NO 直接进入直播间、相同用户将被踢出
 #discussion 目前此参数只对老师起作用、其他角色相当于 `validateConflict` 传 NO
 #discussion 直接调用 `enter` 也相当于 `validateConflict` 传 NO
 */
[self.room enterByValidatingConflict:YES];

// 退出教室
[self.room exit];
```

调用 `enter` 之后，当监听到 `BJLLoadingViewController` 加载成功的回调之后UI可以开始更新布局，以及增加教室内的状态或者行为监听。

9. 上下课

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

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

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。

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

```
/**
 当前登录用户信息
 #discussion BJLiveCore 内部【不】读取此处 `loginUser`
 */
@property (nonatomic, readonly, copy) BJLUser *loginUser;

/**
 当前登录用户是否是主讲人
 #discussion 不支持 KVO
 */
@property (nonatomic, readonly) BOOL loginUserIsPresenter; // NON-KVO
```

```
// BJLRoomInfo
@property (nonatomic, readonly) NSString *ID, *title;                       // 教室 ID、名称
@property (nonatomic, readonly) NSTimeInterval startTimeInterval, endTimeInterval;  // 起止时间
@property (nonatomic, readonly) BJLRoomType roomType;                       // 教室类型
@property (nonatomic, readonly) BJLRoomType roomType; // 直播间类型
@property (nonatomic, readonly) BOOL hasStudentRaise; //有无学生上麦
@property (nonatomic, readonly) BOOL isMockLive; // 是否是伪直播，仅大班课
@property (nonatomic, readonly) BOOL isPushLive; // 是否是推流直播，仅大班课
@property (nonatomic, readonly) BOOL isVideoWall; // 是否是视频墙直播，仅大班课
@property (nonatomic, readonly) BOOL isPureVideo; // 是否是纯视频模板（纯视频不能切换布
```

```
局，视频墙可以切换），仅大班课
@property (nonatomic, readonly) BOOL isLongTerm; // 是否为长期大班课，仅大班课
@property (nonatomic, readonly) BOOL isDoubleCamera; // 是否为双摄像头模板，仅大班课
@property (nonatomic, readonly) BJLRoomGroupType roomGroupType; // 直播间分组类型
@property (nonatomic, readonly) BJLRoomNewGroupType newRoomGroupType; // 新版分组直播间分组类型
```

教室的功能配置信息可以通过 `BJLRoom` 的 `featureConfig` 属性获取，获取时机为进入教室成功（监听到 `enterRoomSuccess` ）之后。

```
// 功能设置
@property (nonatomic, readonly, copy, nullable) BJLFeatureConfig *featureConfig __APPLE_API_UNSTABLE;
```

11. viewModel管理

```
/** 进直播间的 loading 状态，参考 `BJLLoadingVM` */
@property (nonatomic, readonly, nullable) BJLLoadingVM *loadingVM;

/** 直播间信息、状态，用户信息，公告等，参考 `BJLRoomVM` */
@property (nonatomic, readonly, nullable) BJLRoomVM *roomVM;

/** 在线用户，参考 `BJLOnlineUsersVM` */
```

```
@property (nonatomic, readonly, nullable) BJLOnlineUsersVM *onlineUsersVM;

/** 发言申请/处理，参考 `BJLSpeakingRequestVM` */
@property (nonatomic, readonly, nullable) BJLSpeakingRequestVM *speakingRequestVM;

/** 音视频 设置，参考 `BJLMediaVM` */
@property (nonatomic, readonly, nullable) BJLMediaVM *mediaVM;

/** 音视频 采集 - 个人，参考 `BJLRecordingVM` */
@property (nonatomic, readonly, nullable) BJLRecordingVM *recordingVM;
/**
 视频采集视图 - 个人，
 #discussion 参考 `BJLRecordingVM` 的 `recordingVideo`、`inputVideoAspectRatio`
 */
@property (nonatomic, readonly, nullable) UIView *recordingView;

/** 音视频 播放 - 他人，参考 `BJLPlayingVM` */
@property (nonatomic, readonly, nullable) BJLPlayingVM *playingVM;

/** 针对旧版 SDK 的 playingVM 创建的适配层，管理发言用户大班课主摄像头位置的音视频信息 */
@property (nonatomic, readonly, nullable) BJLPlayingAdapterVM *mainPlayingAdapterVM;

/** 针对旧版 SDK 的 playingVM 创建的适配层，管理发言用户大班课扩展摄像头位置的音视频信息 */
@property (nonatomic, readonly, nullable) BJLPlayingAdapterVM *extraPlayingAdapterVM;
```

```
32.
33. /** 课件管理、显示、控制，参考 `BJLDocumentVM` */
34. @property (nonatomic, readonly, nullable) BJLDocumentVM *documentVM;
35.
36. /** 画笔管理 */
37. @property (nonatomic, readonly, nullable) BJLDrawingVM *drawingVM;
38.
39. /**
40.  禁用动画课件
41.  #discussion 是否禁用动画课件由两个参数控制，任意一个值为 YES 就禁止
42.  #discussion 1、由服务端通过 `BJLFeatureConfig` 的 `disablePPTAnimation` 控制
43.  #discussion 2、由上层设置这个 `disablePPTAnimation`
44.  */
45. @property (nonatomic) BOOL disablePPTAnimation;
46.
47. /**
48.  课件、画笔视图
49.  #discussion 尺寸、位置随意设定，需要设置为整数的 pt 值
50.  */
51. @property (nonatomic, readonly, nullable) UIViewController<BJLSlideshowUI> *slideshowViewController;
52.
53. /** 云端录课，参考 `BJLServerRecordingVM` */
54. @property (nonatomic, readonly, nullable) BJLServerRecordingVM *serverRecordingVM;
55.
56. /** 打赏，参考 BJLGiftVM */
```

```
@property (nonatomic, readonly, nullable)
BJLGiftVM *giftVM;

/** 聊天/弹幕，参考 BJLChatVM */
@property (nonatomic, readonly, nullable)
BJLChatVM *chatVM;

/** 直播间内作业管理、显示、控制，参考
`BJLHomeworkVM` */
@property (nonatomic, readonly, nullable)
BJLHomeworkVM *homeworkVM;

/** 直播间内云盘文件管理、显示、控制，参考
`BJLCloudDiskVM` */
@property (nonatomic, readonly, nullable)
BJLCloudDiskVM *cloudDiskVM;

/** 直播带货商品显示、控制，参考 `BJLSellVM` */
@property (nonatomic, readonly, nullable)
BJLSellVM *sellVM;

/** 自习室逻辑 */
@property (nonatomic, readonly, nullable)
BJLStudyRoomVM *studyRoomVM;
```

12 大小班切换

大小班切换场景支持在分组教室、线上双师的教室使用，参考教室信息中的 `BJLRoomNewGroupType` 。这种类型的教室可以让全部用户在大教室内分组互动，也可以支持分组用户在小班教室分组互动，并且能够随时切换两种场景。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。

```
// example：老师或者助教切换大小班
```

```
/**
 线上双师课程切换大小班
 BJLErrorCode_invalidUserRole  错误权限，要求老师或助教权限。
*/
BJLError *error = [self.room.roomVM requestSwitchClass];
```

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..."];
    }
    return YES;
}];
```

13. 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，支持任意用户进入教室后发送自定义信令，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:)
```

```
20.         observer:(BJLMethodObserver)^BOOL(NSString *key, id _Nullable _value, BOOL isCache) {
21.     bjl_strongify(self);
22.     NSLog(@"didReceiveCustomizedBroadcast %@ %@", key, _value);
23. }];
24. 
25. /**
26.  获取定制广播信令缓存
27.  #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:`
28.  #param key     信令类型
29.  #return BJLError:
30.  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
```