

鸿蒙 直播 Core SDK

- 包名: `@baijiayun/livecore` (har 包), 当前版本 `1.9.1`
- **OHPM** 仓
库: `https://ohpm.openharmony.cn/#/cn/detail/@baijiayun%2Flivecore`
- Platform: HarmonyOS NEXT, runtimeOS: HarmonyOS, compatibleSdkVersion: `5.0.0(12)`
- CPU: arm64-v8a
- IDE: **DevEco Studio 5.0** 及以上
- 底层依赖: `brtcohosdk` (音视频引擎)、`@baijiayun/ijkplayer` (播放器)、`@baijiayun/painter` (画笔引擎), 均为传递依赖
- 版本说明: 见本文 §19

1. 简介

百家云鸿蒙直播 SDK 提供 **Core SDK** (`@baijiayun/livecore`) 与 **UI SDK** (`@baijiayun/liveui`) 两部分:

- **UI SDK** 基于 Core SDK 实现, 提供面向教育场景的大班课横版三分屏模板, 包含音视频互动、课件展示与画笔、文字聊天、在线人员、互动玩法等功能, 可快速接入, 适合需要快速上线的场景。UI SDK 开源, 详见《鸿蒙 直播 UI SDK》。
- **Core SDK** 是无 UI 的纯实现库, 涵盖直播间几乎全部功能: 音视频推拉流、信令服务器通信、聊天服务器通信、课



件展示与画笔绘制、教室工具等，适合需要完全自定义界面的场景。

鸿蒙端接口与 *Android 直播 Core SDK* 保持对齐，并做了以下平台适配：

- *Android RxJava Observable* → 鸿蒙端 `LPEmitter<T>`（`observableOfXxx.subscribe(c b)`，见 §17.1）；一问一答类接口返回 *Promise*
- *Android 直接传入 View 渲染* → 鸿蒙端由宿主摆放 *XComponent*、将其 *id* 交给 *SDK* 分配窗位（见 §5.1）
- *Android PPTView* → 鸿蒙端 *WhiteboardPanel* 组件（见 §8）
- *Android LiveSDK.customEnvironmentPrefix* 全局专属域名 → 鸿蒙端在每次进房时通过参数 `domain` 传入

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：上课 / 下课 发布公告、处理举手、 远程开关他人麦克风与摄像头、开关云端录 全体禁言
助教	管理员，拥有部分老师权限， 老师可对助教权限进行管理
学生	听讲人，权限受限，无法管理他人的直播间
教室	直播间。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，老师和助教可以同意或拒绝
发言	发布音频、视频。SDK 层面发言不要求举手

采集	通过设备摄像头、麦克风获取本地音视频数
播放	播放他人发布的音视频，支持同时播放多人
录课	云端录制课程，可生成教室回放
聊天	直播间内的群聊、私聊，支持图片、表情、引用、撤回
课件	课件第一页是白板；老师可上传文档、图片格式的课件，上传成功后可在直播间内支持 PPT 动画
画笔	老师、助教或获得授权的学生可在白板和课件上添清除画笔。 添加画笔的用户当前课件页必须与老师保持
公告	由老师编辑、发布，可包含跳转链接，即时
水印	视频画面上的机构水印，由后台配置下发
点名	老师发起点名，用于签到
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
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
测验	学生收到老师发布的测验并答题
抽奖	标准抽奖与口令抽奖
PK	两名学生对决，其余学生投票
专注度检测	判断学生是否在教室内上课

学情报告	课堂学习情况的统计报告
课后评价	课程结束时对课程与主讲人的评价问卷

1.2 功能描述

1. 教室管理

功能	接口
进入 / 退出教室及相应事件监听	LiveRoom
进房进度、异常监听、断线重连	LiveRoom
老师：上课 / 下课	LiveRoom
公告、全体静音、禁止举手、禁止学生翻页	LiveRoom

2. 在线用户

功能	接口
加载在线用户、监听用户进出教室、在线人数	OnlineUserVM
踢出用户、顶号通知	OnlineUserVM

3. 音视频采集与发布

功能	接口
开启 / 关闭摄像头与麦克风、发布 / 停止发布	LiveRoom 、 LPRecorder
清晰度、镜像、前后摄像头切换、基础美颜	LiveRoom

屏幕共享	LiveRoom 、 LPRecorder
上下行丢包率	LiveRoom

4. 音视频播放

功能	接口
台上用户列表与音视频状态	SpeakQueueVM 、 LPMedia
视频窗位分配、播放 / 关闭指定用户的音视频	LiveRoom
台下观看合流	LPPlayer

5. 举手与发言

功能	接口
学生举手、取消举手，老师处理举手申请	SpeakQueueVM
老师邀请、请下台、远程控制学生音视频	SpeakQueueVM
主讲人切换、学生画笔授权	SpeakQueueVM

6. 课件与画笔

功能	接口
课件显示（静态 / 动画 PPT 自动切换）、翻页、缩放、画笔	WhiteboardPanel
课件列表、上传、删除、多白板	DocListVM
作业区	DocListVM

7. 聊天

功能	接口
发送文字、表情、图片消息，私聊，@，引用	ChatVM
禁言、撤回、置顶、翻译	ChatVM

8. 教室工具

功能	接口
计时器、点名、答题器、抢答器、 网页课件、问答、点赞奖励	ToolBoxVM
抽奖、PK、专注度检测（学生端）	ToolBoxVM
测验（学生端）	QuizVM
红包雨	LiveRoom

9. 其他

功能	接口
商品列表、讲解、 点赞、活动推送	LiveShowVM
云端录制、生成回放	LiveRoom
自定义广播消息	LiveRoom
日志落盘与崩溃采集	LPFileLog 、 LPCrashCollector

2. 快速集成

2.1 包依赖

在模块的 `oh-package.json5` 中添加：

```
1. {  
2.   "dependencies": {  
3.     "@baijiayun/livecore": "^1.9.1"  
4.   }  
5. }
```

执行 `ohpm install` 即可。

音视频引擎 `brtcohosdk`、画笔引擎

`@baijiayun/painter`（及其依赖

`@baijiayun/common`）、播放器 `@baijiayun/ijkplayer`

均由 `@baijiayun/livecore` 传递依赖，无需在 `oh-package.json5` 中额外声明。它们是独立演进的基础库，版本号与 Core SDK 不同属正常现象。

注意：画笔相关类型请从 `@baijiayun/livecore` 导入（见 §8.2），不要直接依赖 `@baijiayun/painter`。

2.2 版本号说明

版本号格式为 `大版本.中版本.小版本`：

- 小版本升级只修复问题，不影响功能与接口；
- 中版本升级会新增功能与接口，个别情况下会调整接口语义，相关变更会在 §19 中给出迁移说明；
- 大版本升级可能包含较大的接口调整。

运行期可通过 `LIVE_SDK_VERSION` 常量读取当前 Core 版本号：

```
1. import { LIVE_SDK_VERSION } from  
   '@baijiayun/livecore';
```

SDK 每次启动会把版本号写入日志首行，反馈问题时附带日志即可定位到具体版本。

2.3 权限声明

在 `module.json5` 中声明以下权限：

权限	用途
<code>ohos.permission.INTERNET</code>	网络
<code>ohos.permission.GET_NETWORK_INFO</code>	进程
<code>ohos.permission.CAMERA</code>	摄像头
<code>ohos.permission.MICROPHONE</code>	麦克风
<code>ohos.permission.KEEP_BACKGROUND_RUNNING</code>	退后台
<code>ohos.permission.VIBRATE</code>	课程提醒
<code>ohos.permission.ACCESS_BLUETOOTH</code>	蓝牙

`CAMERA` 与 `MICROPHONE` 属于 `user_grant` 级权限，需在应用中通过 `abilityAccessCtrl.createAtManager().requestPermissionsFromUser()` 主动向用户申请。建议在进入教室之前完成申请。

2.4 初始化 SDK

Core SDK 无需全局初始化。建议在 `UIAbility.onCreate()` 中尽早开启日志落盘与崩溃采集，便于问题排查：

```
1. import { LPFileLog, LPCrashCollector } from
   '@baijiayun/livecore';
2.
3. export default class EntryAbility extends UIAbility
   {
```

```
4. onCreate(): void {  
5.     LPFileLog.init(this.context); // 先初始化日志目录  
6.     LPCrashCollector.start();    // 再开启崩溃采集  
7. }  
8. onBackground(): void { LPFileLog.flush(); }  
9. onDestroy(): void { LPFileLog.flush(); }  
10. }
```

API	说明
LPFileLog.init(context: common.UIAbilityContext): void	创建日志目录并开始落盘。 请尽早调用， 之前产生的日志不会被记录
LPFileLog.flush(): void	将缓存中的日志写入文件
LPFileLog.write(level: LLogLevel, tag: string, msg: string): void	写入一条日志， 可将应用自身的日志写入同一文件
LPFileLog.getDir(): string	SDK 日志目录
LPFileLog.getBrtcLogDir(): string	音视频引擎日志目录
LPFileLog.listFiles(): string[]	列出全部日志文件
LPFileLog.clear(): void	清空日志目录
LLogLevel	日志级别: Debug / Info / Warn / Error
LPCrashCollector.start(): void	开启崩溃采集，需在 LPFileLog.init() 之后调用。 崩溃信息会在应用下次启动时写入日志目录

日志目录位于应用沙箱 `{context.filesDir}/bjyLogFile/` 下，音视频引擎日志位于其子目录 `log/brtc/`。请使用 `getDir()` / `getBrtcLogDir()` 获取路径，不要自行拼接。

说明：SDK 只负责日志落盘，日志上传由应用自行实现。
实现「上传日志」功能时，将 `getDir()` 目录整体打包上传即可，崩溃文件也在该目录下。

专属域名：SDK 没有全局域名配置，专属域名在每次进房时通过进房参数的 `domain` 字段传入（见 §3.2）。专属域名可在百家云账号中心获取，参考 [专属域名说明](#)。

2.5 调试日志

```
1. import { setWebrtcLogEnabled } from
   '@baijiayun/livecore';
2.
3. setWebrtcLogEnabled(true); // 需在进房前调用，默
   认关闭
```

开启后，音视频引擎的日志会输出到 `hilog`，便于开发阶段排查推拉流问题。正式发布的应用建议保持关闭。

3. 教室管理

3.1 要点说明

- SDK 的全部直播功能都基于「教室」这一场景，**进入教室成功之后**才能正常使用各功能模块。
- 教室状态以服务端为准。例如老师翻页时，SDK 先向服务端发出翻页请求，收到服务端的翻页广播后才真正翻页（老师自己也是如此）。因此界面状态请以各 `observableOfXxx` 事件为准，不要在调用请求接口后自行修改界面状态。

- 各模块的事件均以 `observableOfXxx` 字段提供，类型为 `LPEmitter<T>`（见 §17.1）。订阅后在页面销毁时需要反订阅，本文示例为简洁起见省略了反订阅。

3.2 进入教室

进房分两步：创建 `LiveRoom` 实例、调用进房接口。SDK 提供参加码与签名两种进房方式。

参加码进房

```
1. import { LiveRoom, EnterRoomParams } from
   '@baijiayun/livecore';
2. import common from
   '@ohos.app.ability.common';
3.
4. const room = new LiveRoom(getContext(this) as
   common.UIAbilityContext);
5.
6. // ① 先订阅进房事件
7. room.observableOfLaunchSteps.subscribe((s) =>
   {
8.   const percent: number = Math.floor(s.step * 100
   / s.totalStep);
9. });
10. room.observableOfEnterSuccess.subscribe((result)
   => {
11.   // result.started 是否已上课 / result.userCount 当
   前人数
12. });
13. room.observableOfError.subscribe((e) => {
14.   // e.code: LPErrroCode / e.message
15. });
16.
17. // ② 进房
```

```
18. const params: EnterRoomParams = {
19.   domain: 'https://demo123.at.baijiayun.com', //
    机构专属域名（含 https://）
20.   code: '参加码',
21.   userName: '张三'
22. };
23. room.enterRoom(params);
```

字段	类型	说明
domain	string	机构专属域名，形如 https://demo123.at.baijiayu
code	string	参加码
userName	string	用户昵称

说明：参加码方式下，用户角色、用户编号、头像、分组等信息由服务端根据参加码确定，客户端无需传入。需要自行指定这些信息时，请使用签名进房。

签名进房

适用于已有自己用户体系、不希望用户输入参加码的场景。由业务服务端计算签名 `sign` 后下发给客户端，客户端携带签名直接进房。

```
1. import { LiveRoom, SignEnterRoomParams,
    LPUserType } from '@baijiayun/livecore';
2.
3. const params: SignEnterRoomParams = {
4.   domain: 'https://demo123.at.baijiayun.com',
5.   roomId: '房间号',
6.   userNumber: '10086',
7.   userName: '张三',
8.   userType: LPUserType.Student,
```

```
9.   sign: '业务服务端计算的签名'
10. };
11. room.enterRoomWithSign(params);
```

字段	类型	必填	说明
domain	string	是	机构专属域名
roomId	string	是	房间号
userNumber	string	是	用户编号， 请使用业务系统 定义的用户 ID， 关系到计费统计
userName	string	是	用户昵称
userType	LPUserType	是	用户角色
sign	string	是	签名
userAvatar	string	否	头像地址
groupId	number	否	分组 ID， 不分组时不传
customStr	string	否	自定义透传字符串 房间内其他用户 到

注意： `sign` 必须由业务服务端计算，切勿在客户端计算或存放密钥。计算规则参考 [签名规则](#)。必填参数缺失时， `observableOfError` 会回调 `LPErrCode.InvalidParams`。

账号已在其他端登录

老师账号同一时间只允许在一个终端进入教室。若进房时提示该账号已在其他端登录，可在用户确认后强制进入：

```
1. room.enterRoomIgnoreTeacherUnique();
```

3.3 进房回调说明

字段	说明
<code>observableOfLaunchSteps:</code> <code>LPEmitter<LPLaunchStep></code>	进房进度，可据此实现进房进度回调
<code>observableOfEnterSuccess:</code> <code>LPEmitter<RoomLoginResult></code>	进房成功
<code>observableOfError:</code> <code>LPEmitter<LPRoomError></code>	进房失败及教室运行期间异常，统一从这里回调

```
1. interface LPLaunchStep {
2.     step: number;    // 当前步骤
3.     totalStep: number; // 总步骤数（首次进房与断线重连的步骤数不同）
4.     taskName: string; // 当前步骤名，取值见 LPLaunchTaskName
5. }
6.
7. interface RoomLoginResult {
8.     code: number;    // 0 表示成功
9.     started: boolean; // 是否已上课
10.    userCount: number; // 当前在线人数
11.    linkType: number; // 链路类型
12.    timestamp: number; // 服务端时间戳
13. }
```

进房失败时的错误说明见 §18。

3.4 断线重连

SDK 内置断线自动重连，可通过以下接口监听重连状态，或手动触发重连：

```
1. room.observableOfReconnect.subscribe((state:
    ReconnectState) => {
2.   // Disconnected → Reconnecting → Success /
    Failed
3. });
4. room.reconnect(); // 手动重连
```

ReconnectState : Disconnected （已断开） /
Reconnecting （重连中） / Success （重连成功） /
Failed （重连失败）。

注意：重连成功后不会自动恢复发言，台上用户会回到未发言状态，需要重新走上台流程。

3.5 退出教室

```
1. room.quitRoom(); // 退出教室
2. room.destroy(); // 释放全部资源
```

一般在教室页面的 aboutToDisappear() 中调用。

3.6 功能模块一览

LiveRoom 是整个 SDK 的入口，各功能模块以只读字段提供：

字段	类型	说明
room.speakQueueVM	SpeakQueueVM	举手、发言、主讲人

room.mediaVM	LPMediaVM	台上媒体
room.onlineUserVM	OnlineUserVM	在线用户
room.docListVM	DocListVM	课件、 画笔、作
room.chatVM	ChatVM	聊天
room.toolBoxVM	ToolBoxVM	教室工具
room.quizVM	QuizVM	测验
room.liveShowVM	LiveShowVM	商品与点
room.player	LPPlayer	合流播放
room.getRecorder()	`LPRecorder`	null`

注意：各 VM 上的 `attach()` / `reset()` / `destroy()` 为生命周期方法，由 `LiveRoom` 在进入、重连、退房时自动调用，请勿手动调用。

4. 本地推流（LPRecorder）

4.1 创建推流器

HarmonyOS 的 `XComponent` 不能在父节点之间移动，因此本地预览所用的 `XComponent` 需要在创建推流器时指定。正确顺序为：界面中创建好 `XComponent` → `createRecorder(cfg)` → 之后通过 `getRecorder()` 获取。在 `createRecorder()` 之前，`getRecorder()` 返回 `null`。

```
1. import { LPRecorder, RecorderConfig } from
   '@baijiayun/livecore';
2. 
```

```

3. const cfg: RecorderConfig = {
4.   xcomponentId: 'localPreview', // 承载本地预览的
   XComponent id
5.   width: 640, height: 360, fps: 15, bandwidth:
   400,
6.   brtcApiUrl: '' // 留空即可
7. };
8. const recorder: LPRecorder =
   room.createRecorder(cfg);

```

LPRecorder 主要接口：

API	说明
publish(withVideo: boolean = true): void	发布流
stopPublishing(): void	停止发布
isPublishing(): boolean	是否正在发布
startScreenCapture(spec: LPResolutionSpec): void	开始屏幕共享
stopScreenCapture(): void	停止屏幕共享
isScreenSharing(): boolean	是否正在屏幕共享
getVideoWidth(): number / getVideoHeight(): number	实际采集宽高
destroy(): void	释放推流器
observableOfFirstFrameSent: LPEmitter<boolean>	首帧已发送， 表示画面已成功推出
observableOfReadyToPlay: LPEmitter<boolean>	本地摄像头首帧已就绪， 可用于撤掉本地预览的加 载提示

observableOfError:

LPEmitter<LPRoomError>

推流错误

4.2 开关摄像头与麦克风

```
1. room.attachVideo();    // 打开摄像头
2. room.detachVideo();    // 关闭摄像头
3. room.attachAudio();    // 打开麦克风
4. room.detachAudio();    // 关闭麦克风
5. room.attachAVideo();   // 同时打开音视频
6. room.detachAVideo();   // 同时关闭音视频
7. room.isVideoAttached(); // 是否正在发布视频
8. room.isAudioAttached(); // 是否正在发布音频
9. room.isPublishing();   // 是否正在发布
```

- 发布音视频需要处于**上课状态**；
- 关闭摄像头时会真正释放摄像头设备；
- 在尚未进入音视频房间时调用打开接口，SDK 会记录该意图，并在发布成功后自动生效。

老师上课并发布音视频：

```
1. room.startClassAndPublish(); // 上课成功后自动开始发布
```

注意：请确保应用已获得摄像头、麦克风权限后再打开采集，否则会回调

`LPErrorCode.OpenAudioCameraFailed`。

4.3 清晰度

```
1. import { LPResolutionSpec } from
   '@baijiayun/livecore';
2.
```

```
3. const list: LResolutionSpec[] =  
    room.getResolutionList(); // 当前机构可选的清晰度  
    档位  
4. const def: LResolutionSpec =  
    room.getDefaultResolution(); // 默认档位  
5. room.setCaptureVideoDefinition(def.width,  
    def.height, def.fps, def.bandWidth);
```

LResolutionSpec 字段: type / label / width /
height / fps / bandWidth 。

LResolutionType 档位:

枚举	值	分辨率
LOW	0	320×180
_360	4	480×270
HIGH	1	640×360
_540	5	960×540
_720	2	1280×720
_1080	3	1920×1080

说明：可选档位由机构后台配置决定，帧率与码率由 SDK
根据档位给出。请以 `getResolutionList()` 返回的结果
为准，不要在应用中写死。

4.4 镜像与摄像头切换

```
1. room.setLocalVideoMirrorMode(true); // 本地预览  
    镜像  
2. room.getLocalVideoMirrorMode();
```

```
3. room.switchCamera(true); // true 前置 /
   false 后置
4. room.isFrontCamera();
```

镜像只影响本地预览，不影响其他人看到的画面。

横屏 180° 旋转

若教室页面支持正、反两个横屏方向，屏幕方向变化时需要校正发布出去的画面与本地预览：

屏幕方向	<code>room.setCaptureUpsideDown()</code>	本地预览
正向横屏	<code>true</code>	旋转 180°
反向横屏	<code>false</code>	不旋转

调用时机：屏幕方向变化时，以及**开始发布音视频时**（用户可能在上台前已旋转屏幕）。使用 UI SDK 时无需处理，教室模板已内置。

4.5 基础美颜

```
1. if (room.isBeautySupported()) {
2.   room.setBeautyLevel(0.5); // 磨皮，取值 [0.0, 1.0]
3.   room.setWhitenessLevel(0.5); // 美白，取值 [0.0, 1.0]
4. }
```

部分设备不支持美颜，请先通过 `isBeautySupported()` 判断。

4.6 屏幕共享

```
1. room.startScreenCapture();
2. room.stopScreenCapture();
3. room.isScreenSharing();
4. room.getScreenShareDefinition(); // 机构配置的屏幕共享清晰度
5. room.observableOfScreenShareState.subscribe((s: LPScreenShareState) => { ... });
```

注意：屏幕共享期间应用需持有长时任务（见 §16），否则退到后台后系统会停止采集。

4.7 网络质量

```
1. room.observableOfUpLossRate.subscribe((rate: number) => { ... }); // 上行丢包率 (%)
2. room.observableOfDownLossRate.subscribe((rate: number) => { ... }); // 下行丢包率 (%)
3. const levels: number[] =
    room.getPacketLossRateLevel(); // 机构配置的网络质量分档阈值
```

5. 音视频拉流（LPPlayer）

5.1 视频窗位

HarmonyOS 的 `XComponent` 不能在父节点之间移动，因此 SDK 采用「固定窗位」模型：界面预先摆好若干

`XComponent` 作为窗位，把窗位 id 交给 SDK，由 SDK 决定每个窗位显示谁的画面，界面根据 SDK 给出的分配结果渲染名字、占位图等。

```
1. // ① 界面创建好 XComponent（主视频窗 + 台上若干格）
2. // ② 告诉 SDK 这些窗位，并指定主讲人窗位
3. room.setRemoteRenderViews(['stage0', 'stage1', 'stage2'], 'mainVideo');
4.
5. // ③ 本地画面所在的窗位（如老师端显示在主视频窗）
6. room.setLocalRenderSlot('mainVideo');
7.
8. // ④ 订阅窗位分配结果
9. room.observableOfRemoteLayout.subscribe((slots: RemoteSlotInfo[]) => {
10. // 按 slotId 找到对应的 XComponent，渲染名字条、占位图等
11. });
12. const current: RemoteSlotInfo[] = room.getRemoteLayout();
```

RemoteSlotInfo :

字段	类型	说明
slotId	string	窗位 id，即传给 setRemoteRenderViews id
userId / userNumber / userName	string	当前占用该窗位的用户；u 为空表示窗位空闲
userType	number	用户角色
videoOn	boolean	该窗位当前是否有画面（无画面时应显示占位图）
audioOn	boolean	麦克风是否打开

mediaId	string	该窗位当前显示的流标识
sourceType	number	流类型，见 <code>MediaSource</code>

注意：`videoOn` 为 `false` 可能是对方关闭了摄像头，也可能是本端主动关闭了对方的视频。判断本端是否关闭了某人的视频，请使用 `isVideoClosedByUser(userId)`。

配套接口：

API	说明
<code>swapWithPresenterSlot(userId: string): void</code>	将某人与主讲人
<code>switchLocalPreviewTo(xcomponentId: string): void</code>	将本地预览切换 <code>XComponent</code>

注意：只需调整本地画面的显示位置或大小时，请直接修改该 `XComponent` 的位置与尺寸，**不要使用** `switchLocalPreviewTo`。切换承载组件会重建渲染表面，可能导致画面短暂中断。

5.2 台上用户

```
1. room.speakQueueVM.observableOfActiveUsers.subscribe(
    PublishedMedia[] => { ... });
2. const list: PublishedMedia[] =
    room.speakQueueVM.getActiveUserList();
```

`PublishedMedia`：

字段	类型	说明
<code>userId</code> /	<code>string</code>	用户信息

userNumber / userName		
userType	number	用户角色
videoOn / audioOn	boolean	视频 / 音频是否打开
width / height	number	推流分辨率
isScreenSharing	boolean	是否为屏幕共享流
isSelf	boolean	是否为自己
mediald	string	该条记录的唯一标识
realMediald	string	该流在音视频引擎中的、与 RemoteSlotInfo.medi 对应
sourceType	number	流类型，见 MediaSourceType

MediaSourceType : MainCamera (0) 摄像头 / ExtCamera (1) 副摄像头 / MainScreenShare (2) 屏幕共享 / ExtScreenShare (3) 副屏幕共享 / Media (4) 媒体课件 / ExtMedia (5) 副媒体课件。

注意：同一用户可能同时发布多路流（例如老师同时发布摄像头与屏幕共享），此时列表中会出现 userId 相同、mediald 不同的多条记录。

- 只关心「人」（如名单、台上人数）时，请先过滤 sourceType === MediaSourceType.MainCamera ；
- 以列表元素为键做缓存时，请使用 mediald 而非 userId ；

- 与窗位比对时，请使用 `PublishedMedia.realMediaId` 与 `RemoteSlotInfo.mediaId` 比对。

LPMediaVM (room.mediaVM) :

API	说明
<code>observableOfActiveUsers: LPEmitter<PublishedMedia[]></code>	台上媒体列表
<code>observableOfSelfPublished: LPEmitter<boolean></code>	自己是否正在发布
<code>observableOfMixStreamReady: LPEmitter<boolean></code>	合流是否就绪
<code>getActiveUsers(): PublishedMedia[]</code>	台上媒体列表
<code>hasActiveVideo(): boolean</code>	台上是否有人打开了摄
<code>isPublished(): boolean</code>	自己是否已发布

5.3 本端观看控制

以下接口只影响本端观看，不影响对方发布，也不影响其他人观看：

- `room.playVideo(userId);` // 观看某人的音视频
- `room.playAudio(userId);` // 只听声音，不看画面
- `room.playAVClose(userId);` // 停止观看某人的音视频
- `room.getRemotePlayMode(userId);` // 当前观看模式: `RemotePlayMode.Both / AudioOnly / None`
- `room.isVideoClosedByUser(userId);` // 本端是否关闭了某人的画面

6. `room.muteRemoteAudio(userId, true);` // 静音某个远端用户
7. `room.muteRemoteVideo(userId, true);` // 停止渲染某个远端用户的画面

台下观看合流时，传入

`LiveRoom.FAKE_MIX_STREAM_USER_ID` 控制合流画面：

1. `room.playAudio(LiveRoom.FAKE_MIX_STREAM_USER_ID);`
// 合流只听声音
2. `room.playVideo(LiveRoom.FAKE_MIX_STREAM_USER_ID);`
// 恢复合流画面

说明：老师远程控制学生的摄像头、麦克风请使用 §6.2 的 `controlRemoteUser`，与本节的本端观看控制是两回事。

5.4 台下观看合流（LPPlayer）

大班课中，台下学生观看的是所有台上用户合成的一路**合流**；学生上台后，SDK 自动切换为实时互动模式。学生端可开启自动处理：

1. `room.autoPullMixStreamWhenActive();` // 台上有人发布时自动播放合流
2. `room.autoSwitchWebRTCModeWhenOnStage(stageC);` // 上台 / 下台时自动切换观看模式

两个方法均可在 `enterRoom()` 之前调用，仅对学生端生效。

承载合流的 `XComponent` 需按如下方式声明：

1. `XComponent({`

- ```
2. id: 'mixStream', type:
 XComponentType.SURFACE,
3. libraryname: 'ijkplayer_napi', controller:
 this.xcController
4. })
5. .onLoad((event?: object) => {
 room.player.setNativeContext(event); })
```

LPPlayer ( room.player ) :

| API                                                   | 说明                                                  |
|-------------------------------------------------------|-----------------------------------------------------|
| `setNativeContext(ctx: object \                       | undefined): void`                                   |
| playMixStream(): void                                 | 播放合流                                                |
| stop(): void / isPlaying(): boolean / destroy(): void | 播放控制                                                |
| observableOfPlaying: LPEmitter<boolean>               | 播放状态                                                |
| observableOfMixVideoRatio: LPEmitter<number>          | 合流画面宽高比，可据此i                                        |
| observableOfReadyToPlay: LPEmitter<string>            | 某一路画面已就绪，参数为RemoteSlotInfo.mediaLiveRoom.FAKE_MIX_S |

### 加载提示

开始播放前显示加载提示，收到 observableOfReadyToPlay 后撤掉：

- ```
1. room.player.observableOfReadyToPlay.subscribe((m
  string) => {
```

2. `// 按 mediaId 找到对应窗位，撤掉加载提示`
3. `});`

说明：同一路流停止后再次播放会再次回调。若界面自行维护「已就绪」集合，切换观看模式时请清空。

6. 发言管理 (SpeakQueueVM)

6.1 学生端

```
1. const vm = room.speakQueueVM;
2.
3. vm.requestSpeakApply();    // 举手申请上台
4. vm.cancelSpeakApply();    // 取消举手
5. vm.leaveStage();          // 主动下台
6. vm.sendSpeakInvite(true); // 响应老师的上台邀请
   (true 同意 / false 拒绝)
7.
8. vm.observableOfApplyCountDown.subscribe((sec:
   number) => { ... }); // 举手倒计时
9. vm.observableOfApplyTimeout.subscribe(() => {
   ... }); // 举手超时
10. vm.observableOfSpeakApplyDeny.subscribe(()
   => { ... }); // 举手被拒绝
11. vm.observableOfSpeakInvite.subscribe(() => { ...
   }); // 收到上台邀请
12. vm.observableOfMediaControl.subscribe((c:
   MediaControlCommand) => { ... }); // 老师远程控制我的音视频
13. vm.observableOfSpeakClosedByTeacher.subscribe(
   => { ... }); // 被老师请下台
```

```
14. vm.observableOfSelfSpeaking.subscribe((on:
    boolean) => { ... }); // 自己是否在台上
15. vm.isSelfSpeaking();
```

6.2 老师端

```
1. vm.observableOfApplyList.subscribe((list:
    SpeakApplyUser[] => { ... }); // 举手列表
2. vm.getApplyList();
3.
4. vm.agreeSpeakApply(userId); // 同意举手
5. vm.disagreeSpeakApply(userId); // 拒绝举手
6. vm.closeOtherSpeak(userId); // 关闭某人发言
7. vm.kickOffStage(user); // 请某人下台
8. vm.controlRemoteUser(userId, videoOn,
    audioOn); // 远程控制学生的摄像头、麦克风
9.
10. vm.observableOfSpeakApplyResResult.subscribe((r:
    SpeakApplyResult) => { ... }); // 处理举手的结果
```

SpeakApplyUser : userId / userNumber /
userName / userType 。

SpeakApplyResult : userId / agreed / isSelf 。

MediaControlCommand : videoOn / audioOn 。

说明：同意举手时若台上人数已达上限，处理结果同样通过
observableOfSpeakApplyResResult 告知。

6.3 主讲人

```
1. vm.getPresenter(); // 当前主讲人
    userId
2. vm.requestSwitchPresenter(userId); // 切换主讲
    人
```

```
3. vm.observableOfPresenterChange.subscribe((userInfo:
    string) => { ... });
4. room.isSwitchPresenterEnabled(); // 机构是否允许切换主讲
```

6.4 学生画笔授权

```
1. vm.requestStudentDrawingAuth(userNumber,
    grant); // 授予 / 收回某学生的画笔权限
2. vm.hasDrawingAuth(userNumber); // 某学生是否有画笔权限
3. vm.getStudentsDrawingAuthSet(); // 全部已授权学生的 userNumber
4. vm.getDrawingAuth(); // 自己是否有画笔权限
5. vm.observableOfDrawingAuth.subscribe((has:
    boolean) => { ... });
6. vm.observableOfStudentDrawingAuth.subscribe((userInfo:
    string[]) => { ... });
```

注意：授权名单以 `userNumber` 记录（`userId` 在重连后可能变化）。学生上台时会自动获得画笔权限。

7. 在线用户（OnlineUserVM）

```
1. const vm = room.onlineUserVM;
2.
3. vm.observableOfOnlineUser.subscribe((list:
    OnlineUser[]) => { ... }); // 在线用户列表
4. vm.observableOfOnLineUserCount.subscribe((count:
    UserCountInfo) => { ... }); // 在线人数
```

```

5. vm.observableOfUserIn.subscribe((u: OnlineUser)
=> { ... });           // 用户进入
6. vm.observableOfUserOut.subscribe((u:
OnlineUser) => { ... }); // 用户离开
7. vm.observableOfKickOut.subscribe((k:
KickOutInfo) => { ... }); // 自己被请出教室
8. vm.observableOfLoginConflict.subscribe((c:
LoginConflictInfo) => { ... }); // 账号在其他端登录
9.
10. vm.getOnlineUserList();   vm.getUserCount();
11. vm.getUserById(userId);
    vm.getUserByNumber(userNumber);
12. vm.hasMoreUser();        vm.loadMoreUser();
13. vm.requestKickOutUser(userId, true); // 请出教
    室，第二个参数表示是否禁止再次进入

```

类型	字段
OnlineUser	userId / userNumber / userName / userType / avatar / endType / groupId / isSelf
UserCountInfo	count 当前在线人数 / accumulative 累计人数
KickOutInfo	code / reason / notBlock
LoginConflictInfo	endType / endTypeName （在哪个终端登录）

分页加载：教室人数较多时，服务端不再逐个推送用户进出事件，在线用户列表需通过 `loadMoreUser()` 分页加载，`hasMoreUser()` 表示是否还有下一页。

说明：查找教室中的老师或当前主讲人，请使用
`room.getTeacherUser()` /

`room.getPresenterUser()`，不要遍历在线用户列表
(列表为分页加载，不保证包含全部用户)。

设备状态上报：当摄像头 / 麦克风未授权、被占用或应用进入后台时，可上报设备状态，供老师端展示：

```
1. import { MediaState } from '@baijiayun/livecore';
2. vm.updateMediaState(MediaState.PermissionDeny, null);
```

`MediaState` : `Normal` (0) 正常 / `Unavailable` (1) 不可用 / `PermissionDeny` (2) 未授权 / `Occupied` (3) 被占用 / `Backstage` (4) 在后台。

8. 课件与画笔

Core SDK 提供以下 ArkUI 组件承载课件与画笔：

组件	用途
<code>WhiteboardPanel</code>	推荐使用。 静态 / 动画 PPT 自动切换，内置翻页、缩放、画笔与加载失败处理
<code>WhiteboardView</code>	仅画笔画布，适用于自建课件外框的场景
<code>AnimPptView</code>	动画 PPT 容器，适用于深度自定义的场景
<code>LaserOverlay</code>	激光笔浮层

8.1 使用 WhiteboardPanel

```
1. import { WhiteboardPanel, PainterGLType,
   shapeJsonData } from '@baijiayun/livecore';
2.
3. WhiteboardPanel({
4.   docListVM: this.room.docListVM,
5.   canPaint: () => this.penMode,           // 当前
   是否为画笔模式
6.   isLaserMode: () => this.laserMode,      // 当
   前是否为激光笔模式
7.   pptTouchMode: this.pptTouchMode,       // 是
   否将触摸交给动画 PPT 页面
8.   onPainterReady: (p: PainterGLType) => {
   this.painter = p; },
9.   onLocalShapeAdd: (s: shapeJsonData,
   hasAppend: boolean) => {
10.    this.room.docListVM.requestShapeAdd(s as
   Record<string, Object>, hasAppend);
11.  },
12.   onLocalShapeAppend: (s) => { ... },
13.   onLocalShapeRemove: (arr) => { ... },
14.   onLocalShapeUpdate: (arr) => { ... },
15.   onLocalTextCommit: (s) => { ... },
16.   onLaserMove: (x, y) => {
   this.room.docListVM.requestShapeLaserUpdate(x,
   y); },
17.   onLaserEnd: () => {
   this.room.docListVM.clearLaser(); },
18.   requestTextInput: async (def, onChanging) => {
   return await this.showTextDialog(def,
   onChanging); },
19.   onDocChange: (page: DocPageModel) => { ...
   },
20.   swipeEnabled: () => true,
21.   animUrlProvider: () =>
   this.room.getAnimPptUrl(),
```

```
22.   animRoomProvider: () => this.room,
23.   onPPTError: (code: number, msg: string, reload:
      () => void) => { ... },
24.   onAnimPPTLoadFinish: () => { ... }
25. })
```

主要参数说明：

参数	说明
canPaint	当前是否为画笔模式
isLaserMode	当前是否为激光笔模式
pptTouchMode	开启后触摸事件直接交给动画 PPT 页面，可操作课件内的可交互元素
swipeEnabled	是否允许滑动翻页
requestTextInput	文字画笔的输入框，由应用弹出输入框并返回输入结果 (Promise<string>)，输入过程中通过 onChanging 回传内容以便实时预览
onPPTError / onAnimPPTLoadFinish	动画课件加载异常 / 加载完成，见 §8.4

注意：WhiteboardPanel 的入参变化会触发组件重新构建。请只传入稳定的值或函数，避免频繁变化的参数导致画布重建。

8.2 画笔类型

画笔引擎的相关类型由 Core SDK 统一导出：

- ```
1. import { PainterGL, PainterGLType, PainterConfig,
 shapeJsonData, shapeJsonPoint, AddShape }
2. from '@baijiayun/livecore';
```

支持 12 种画笔：任意曲线、文字、激光笔、直线、单箭头、双箭头、空心 / 实心矩形、空心 / 实心椭圆、空心 / 实心三角形。

画笔权限请以 SDK 的判断为准，自绘白板时不要自行按角色判断：

- ```
1. room.canOperateShape(shapeRole); // 是否可选中 / 移动该笔迹
2. room.canEraseShape(shapeRole); // 是否可擦除该笔迹
```

8.3 画笔接口

使用 `WhiteboardPanel` 时，画笔的收发已由组件与以下接口配合完成；自建白板时可直接使用：

API	说明
<code>requestPageAllShape(docId, page)</code>	拉取某页的全部笔迹
<code>requestShapeAdd(shape, hasAppend = false)</code>	新增笔迹
<code>requestShapeAppend(shape)</code>	续写笔迹
<code>requestShapeUpdate(shape)</code>	更新笔迹
<code>eraseShape(shapeIds)</code>	擦除指定笔迹
<code>eraseAllShape()</code>	清空当前页笔迹
<code>requestShapeLaserUpdate(x,</code>	更新激光笔位置

y, cursor = false, color = '#FF0000')	
clearLaser()	清除激光笔
observableOfLasers: LPEmitter<LaserInfo[]>	激光笔位置

8.4 动画 PPT

使用 `WhiteboardPanel` 时，SDK 会根据课件类型与机构配置自动选择静态或动画方式显示，并内置加载失败处理：

情况	SDK 行为	回调
动画课件加载失败	自动切换为静态课件	<code>onPPTError(C...)</code>
动画课件加载超时	不自动切换， 由应用决定	<code>onPPTError(C...)</code>
动画课件加载完成	—	<code>onAnimPPTLo</code>

```
1. WhiteboardPanel({
2.   // ...
3.   onPPTError: (code: number, msg: string, reload:
4.     () => void) => {
5.     if (code ===
6.       CODE_ERROR_ANIMPPT_LOAD_TIMEOUT) {
7.       // 提示用户：重新加载 → reload(); 跳过动画 →
8.       room.docListVM.setAnimPptEnable(false)
9.     }
10.  },
11.  onAnimPPTLoadFinish: () => { /* 关闭提示 */ }
12. })
```

相关接口：

1. `room.docListVM.shouldUseAnimPpt();` // 当前是否使用动画方式显示
2. `room.docListVM.isContainH5PPTDoc();` // 是否包含仅支持动画方式的课件
3. `room.docListVM.setAnimPptEnable(false);` // 关闭动画，使用静态方式显示（默认开启）

说明：部分课件仅提供动画版本，对这类课件调用

`setAnimPptEnable(false)` 不生效，加载超时也不会触发 `onPPTError`。断线重连后动画开关会恢复为默认开启。

直接使用 `AnimPptView` 的场景：需要自行处理加载结果与失败，请务必接入 `onLoadResult` / `onLoadError` / `isH5Doc` 三个参数，否则课件加载失败时会一直显示空白。翻页通过 `AnimPptCommander` 控制：

```
1. import { AnimPptView, AnimPptCommander }
   from '@baijiayun/livecore';
2.
3. const commander = new AnimPptCommander();
4. AnimPptView({
5.   room: this.room, url: this.room.getAnimPptUrl(),
   commander: commander,
6.   isH5Doc:
   this.room.docListVM.isContainH5PPTDoc(),
7.   onLoadResult: (ok, detail) => { ... },
8.   onLoadError: (code, message) => { ... },
9.   onPageChangeEnd: (docId, page, step) => { ...
   }
10. })
11.
12. commander.nextStep(); commander.prevStep();
```

```
13. commander.nextPage(); commander.prevPage();
14. commander.gotoPage(docId, page);
15. commander.reload();
```

8.5 页码与翻页

```
1. const vm = room.docListVM;
2.
3. vm.observableOfPageChange.subscribe((p:
  PageChangeInfo) => { ... }); // docId / page /
  step
4. vm.observableOfCurrentDoc.subscribe((d:
  DocPageModel) => { ... });
5. vm.observableOfPageFlipAuth.subscribe((canFlip:
  boolean) => { ... }); // 自己当前是否可以翻页
6.
7. vm.getCurrentDocId(); vm.getCurrentPage();
  vm.getCurrentStep();
8. vm.turnPage(1);      vm.canFlip(1);      //
  相对翻页
9. vm.changeToFlatPage(index);              //
  跳转到第 index 页（全部课件连续编号）
10. vm.getCurrentFlatIndex();
    vm.getMaxFlatIndex();
11. vm.isEnableChangePPTPage();
12. vm.setDisableOverTeacherMaxPage(true);
```

大班课默认将全部课件连续显示， `changeToFlatPage` / `getCurrentFlatIndex` 等接口使用跨课件的连续页码。

学生翻页范围：默认情况下，学生只能翻到老师已经翻到过的页，往回翻不受限制，可翻到的最大页为 `getMaxFlatIndex()`。如需允许学生自由翻页（仅本端生效），进房后调用：

```
1. vm.setDisableOverTeacherMaxPage(false);
```

老师可通过 `room.requestForbidStudentSwitchPPT()` 禁止学生翻页，也可通过 `requestStudentPPTAuthChange()` 单独授予某位学生课件操作权限：

```
1. vm.requestStudentPPTAuthChange(grantAuth,
    userNumber);
2. vm.getStudentPPTAuthList();
3. vm.observableOfStudentPPTAuth.subscribe((number,
    string[]) => { ... });
```

8.6 缩放

```
1. vm.observableOfDocUpdate.subscribe((z) => {
    ... }); // 缩放与滚动位置
2. vm.getDocZoom(docId);
3. vm.requestDocUpdate(zoom, page, step);
```

是否向其他人同步缩放状态由机构配置决定。

8.7 多白板

```
1. vm.addWhiteboardPage();
2. vm.deleteWhiteboardPage(pageId);
```

9. 文件管理（DocListVM）

9.1 课件列表

```
1. vm.requestDocAllReq();
2. vm.observableOfDocListChanged.subscribe((list:
  DocPageModel[]) => { ... });
3. vm.getDocList();
```

使用 `WhiteboardPanel` 时组件已自动拉取课件列表，一般无需再调用 `requestDocAllReq()`。

`DocPageModel` 主要字段：`docId` / `name` / `ext` / `page` / `url` / `width` / `height` / `totalPage` / `isH5Doc` / `pageId` / `size` / `userName`。

9.2 上传与删除课件

上传、转码、添加到教室三个步骤合并为一个异步接口：

```
1. const docId: string = await
  vm.uploadDocument(filePath, isAnimPPT, (stage,
    percent) => {
2.   // stage: 'upload' 上传 / 'transfer' 转码 / 'add' 添
    加; percent: 0~100
3. });
4. vm.deleteDocument(docId);
```

图片也通过 `uploadDocument` 上传。课件的增加与删除统一通过 `observableOfDocListChanged` 通知。

9.3 作业区

```
1. vm.requestHomeworkAllList(keyword); //
  keyword 为空表示全部
2. vm.observableOfHomeworkListChanged.subscribe((
  HomeworkItem[]) => { ... });
3. vm.getHomeworks();
```

```

4.
5. vm.getAllowUploadHomework();           // 是否允许学生提交作业
6. vm.requestUpdateAllowUploadHomework(true);
   // 老师设置是否允许
7. vm.observableOfAllowUploadHomework.subscribe((
   boolean) => { ... });
8.
9. const id: string = await
   vm.uploadHomework(filePath, (stage, percent)
   => { ... });
10. const docId: string = await
   vm.syncHomeworkToClass(item); // 将作业添加为课件
11. const path: string = await
   vm.downloadHomework(item, (p) => { ... });
12. vm.deleteHomework(homeworkId);

```

HomeworkItem : homeworkId / fileId / name /
 ext / size / userName / userNumber /
 userRole / lastModified / canPreview 。

作业功能是否开启: room.isHomeworkEnabled() 。

10. 聊天（ChatVM）

10.1 收发消息

```

1. const vm = room.chatVM;
2.
3. vm.observableOfNotifyDataChange.subscribe((msg
   ChatMessage[]) => { ... }); // 消息列表变化

```

```
4. vm.observableOfReceiveMessage.subscribe((m:
    ChatMessage) => { ... });          // 收到新消息
5. vm.getMessageCount();
6. vm.getMessage(index);                // 越界时返回
    null
7.
8. vm.sendMessage(content);             //
    发送消息
9. vm.sendMessage(content, atList);     //
    @ 他人
10. vm.sendMessage(content, atList,
    referenceMessage); // 引用消息
11. vm.sendEmojiMessage(key);           //
    发送表情，key 来自 getExpressions()
12. vm.sendMessageToUser(content, toUserId,
    toUserNumber, toUserName); // 私聊
```

说明：消息较多时，列表请使用 `getMessageCount()`
+ `getMessage(index)` 按需读
取。`getMessages()` 每次返回完整列表的副本，不适
合在长列表中逐条调用。

`ChatMessage` 主要字段：

字段	说明
<code>id</code>	消息 ID
<code>content</code>	消息内容
<code>fromId</code> / <code>fromNumber</code> / <code>fromName</code> / <code>fromAvatar</code> / <code>fromType</code>	发送者信息
<code>isWhisper</code> / <code>to</code>	是否私聊 / 私聊对象
<code>isSelf</code>	是否自己发送

timestamp	发送时间
atList	被 @ 的用户
reference	被引用的消息

10.2 图片与表情

```
1. // 图片：先上传，再发送
2. const r: UploadDocResult = await
   vm.uploadChatImage(filePath);
3. vm.sendImageMessage(r.url, r.width, r.height);
4. room.canSendChatImage(); // 机构是否允许发送图
   片
5.
6. // 表情
7. const list: ChatExpression[] =
   vm.getExpressions(); // key / name / url
8. vm.sendEmojiMessage(list[0].key);
```

10.3 禁言

```
1. vm.requestForbidAllChat(true); // 全体禁言（老师
   / 助教）
2. vm.observableOfForbidAllChatStatus.subscribe((on:
   boolean) => { ... });
3. vm.isForbidAllChat();
4.
5. vm.forbidChat(userId, userNumber, userName,
   userType, true); // 禁言某人
6. vm.observableOfForbidList.subscribe((ids:
   string[]) => { ... });
7. vm.isUserForbidden(userId);
```

```
8. vm.requestForbidList(); // 重新拉取禁言名单
   (老师 / 助教)
9.
10. vm.observableOfIsSelfChatForbid.subscribe((forbid:
    boolean) => { ... });
11. vm.isChatForbidden(); // 自己是否被禁言
```

被禁言时发送消息，`observableOfError` 会回调
`LPErrorCode.MessageSendForbid`。

10.4 撤回

```
1. vm.requestMsgRevoke(msgId, fromUserId);
2. vm.getRecallState(msg); // ChatRecallState:
   None / Recall / Delete
```

撤回结果已更新在消息列表中（通过
`observableOfNotifyDataChange` 通知），可通过
`getRecallState()` 判断某条消息的撤回状态。

10.5 消息置顶

```
1. vm.observableOfMsgStickyList.subscribe((list:
    ChatMessage[]) => { ... }); // 空数组表示无置顶
2. vm.getMsgStickyList();
3. vm.requestMsgStickyList(messages); // 设置置顶
   列表（老师 / 助教），传空数组表示全部取消
```

`requestMsgStickyList` 每次传入的是完整的置顶列表。取消其中一条时，传入剩余的消息即可。

10.6 消息翻译

```
1. room.isChatTranslationEnabled(); // 机构是否
   开启翻译
2. room.getDefaultTranslateLanguage(); // 老师 /
   助教的默认目标语言
3. room.getDefaultTranslateLanguageStu(); // 学生的
   默认目标语言
4.
5. vm.sendTranslateMessage(text, key, classId,
   userId, 'auto', 'en');
6. vm.observableOfReceiveTranslateMessage.subscribe
   (LPMessageTranslateModel) => {
7.   if (m.code === 0) { /* m.result 为译文 */ }
8. });
```

参数	说明
message	待翻译内容（建议去掉表情标记）
messageId	业务自定义的标识， 翻译结果中原样带回，用于匹配结果
classId / userId	房间号 / 自己的 userId
fromLanguage	源语言，传 'auto' 自动识别
toLanguage	目标语言

目标语言取值：zh 中文（简体） / cht 中文（繁体） /
en 英语 / jp 日语 / vie 越南语 / id 印尼语 /
hkm 高棉语。

注意：1 秒内的重复调用会被忽略，请在界面上自行处理超时提示。翻译相关配置需在进房成功后读取。

10.7 聊天连接状态

```
1. vm.observableOfConnected.subscribe((ok:
  boolean) => { ... });
2. vm.isConnected();
```

聊天使用独立连接，聊天连接异常不影响上课。

11. 教室工具（ToolBoxVM）

```
1. const vm = room.toolBoxVM;
2. const e: ToolboxEnable =
  room.getToolboxEnable(); // 机构开启了哪些工具
```

说明：除特别注明外，以下工具的发起端与参与端接口均已提供。抽奖、PK、专注度检测、测验由老师在 Web 端发起，客户端提供学生端能力。

11.1 计时器

```
1. vm.requestBJTimerStart(current, total,
  countDown);
2. vm.requestBJTimerPause(current, total,
  countDown);
3. vm.requestBJTimerEnd();
4. vm.observableOfTimer.subscribe((t: BJTimerInfo)
  => { ... }); // t.action: TimerAction.Start / Pause
  / Continue / Stop
```

服务端只在开始、暂停、结束时同步状态，界面上每秒的读数需由应用根据 `BJTimerInfo` 自行推算；`isCache` 为 `true` 表示是进房时补发的进行中状态。

11.2 点名

1. `vm.startRollCall(duration);` // 老师发起, `duration` 为时长 (秒)
2. `vm.responseRollCall();` // 学生答到
3. `vm.observableOfRollCallStart.subscribe((s: RollCallStart) => { ... });`
4. `vm.observableOfRollCallResult.subscribe((r: RollCallResult) => { ... });` // `ackList` 已答到 / `nackList` 未答到
5. `vm.observableOfRollCallTimeout.subscribe(() => { ... });`

11.3 答题器

1. `vm.requestAnswerStart(isJudgement, durationSec, description, options, showAnswer);`
// 老师发起
2. `vm.requestAnswerEnd(isRevoke, delay);`
// 老师结束
3. `vm.submitAnswers(sheet, picked, timeUsed);`
// 学生提交
4. `vm.observableOfAnswerStart.subscribe((s: AnswerSheet) => { ... });`
5. `vm.observableOfAnswerUpdate.subscribe((s: AnswerSheet) => { ... });`
6. `vm.observableOfAnswerEnd.subscribe((e: AnswerEnd) => { ... });`

`AnswerSheet` 主要字段: `id` / `type` (0 选择题、1 判断题) / `isVote` (是否投票题) / `duration` / `description` / `options` / `showAnswer`。

11.4 抢答器

```
1. vm.requestAnswerRacerStart(prepareTime); // 老师发起
2. vm.requestAnswerRacerEnd(isRevoke); // 老师结束
3. vm.requestAnswerRacerParticipate(); // 学生抢答
4. vm.observableOfAnswerRacerStart.subscribe((s: RacerStart) => { ... });
5. vm.observableOfAnswerRacerEnd.subscribe((e: RacerEnd) => { ... }); // winnerName / winnerNumber 等
```

11.5 红包雨

红包雨相关接口在 `LiveRoom` 上:

```
1. await room.publishRedPacket(count, amount, timeLength); // 老师发红包
2. room.requestRedPacket(id, duration); // 开始红包雨
3. room.observableOfRedPacket.subscribe((r: RedPacketInfo) => { ... });
4. room.observableOfRedPacketFinish.subscribe((id: string) => { ... });
5. const got: number = await room.robRedPacket(packetId); // 学生抢红包
6. room.observableOfRedPacketRankList.subscribe((to RedPacketTopList) => { ... }); // 排行榜
7. room.getMaxRedPacketCount();
```

说明：排行榜可能先于红包雨结束事件到达，建议收到后暂存，在红包雨结束后再展示；新一轮红包雨开始时请清空上一轮的排行榜。

11.6 网页课件

```
1. vm.requestOpenWebPage(url, status,  
    allowOperate);  
2. vm.observableOfWebPageInfo.subscribe((w:  
    WebPageInfo) => { ... }); // status / url /  
    allowOperate
```

11.7 问答

```
1. vm.requestQuestionSend(content); //  
    学生提问  
2. vm.requestQuestionPullReq(page, status,  
    onlySelf); // 拉取问答列表  
3. vm.requestQuestionReply(id, content);  
    // 回复  
4. vm.requestQuestionPub(id, publish); //  
    发布 / 取消发布  
5. vm.observableOfQuestionPullRes.subscribe((items:  
    QuestionItem[]) => { ... });  
6. vm.observableOfQuestionChanged.subscribe((c:  
    QuestionChanged) => { ... });  
7. vm.observableOfQuestionForbidStatus.subscribe((f:  
    boolean) => { ... });  
8. room.isQaEnabled();
```

11.8 点赞奖励

```
1. vm.requestAward(toUserNumber, userName,  
    awardType); // 奖励某学生  
2. vm.getAwardConfigs(); // 机构  
    配置的奖励类型
```

```

3. vm.getAwardCount(userNumber);
   // 某人累计获得的奖励数
4. vm.observableOfAward.subscribe((a: AwardInfo)
   => { ... });
5. room.isShowOthersAwardEnabled();

```

11.9 抽奖（学生端）

```

1. vm.observableOfLottery.subscribe((r:
   LotteryResult) => { ... }); // 开奖结果
2. vm.observableOfCommandLotteryStart.subscribe((c:
   CommandLottery) => { ... }); // 口令抽奖开始
3. vm.requestCommandLottery(userNumber);
   // 参与口令抽奖
4. vm.sendLotteryResult(beginTime, name, mobile,
   address); // 中奖后提交联系方式

```

说明：仅学生参与抽奖。判断是否中奖请使用中奖名单中的 `userNumber`。口令抽奖中「聊天内容是否与口令一致」由应用判断，一致时调用 `requestCommandLottery()` 参与。

11.10 PK（学生端）

```

1. vm.observableOfPKStatus.subscribe((info:
   PKStatusInfo) => { ... }); // PK 状态
2. vm.observableOfPKVote.subscribe((users:
   PKVoteUser[]) => { ... }); // 票数更新
3. vm.requestPKVote(userNumber);
   // 投票

```

`LPPKStatus`： `PK_END` (0) 已结束 / `PK_START` (1) PK 中 / `PK_VOTE` (2) 投票中。参与 PK 的两名学生不能投票；已

投票的学生可以改投另一方。票数为定时同步，投票后建议在界面上先行更新。

11.11 专注度检测（学生端）

```
1. vm.observableOfAttentionDetection.subscribe(()  
    => {  
2.   vm.requestAttentionReport(applsForeground);  
   // 收到检测时上报应用是否在前台  
3. });  
4. vm.observableOfAttentionAlert.subscribe((content:  
    string) => { ... }); // 老师的提醒内容  
5. room.observableOfAppForeground.subscribe((fg:  
    boolean) => { applsForeground = fg; });
```

11.12 测验（QuizVM，学生端）

测验的答题界面为网页，SDK 负责测验的开始、结束、公布答案等通知与提交：

```
1. const quiz = room.quizVM;  
2. quiz.observableOfQuizStart.subscribe((e:  
    QuizEvent) => { ... }); // 老师发布测验  
3. quiz.observableOfQuizRes.subscribe((e:  
    QuizEvent) => { ... }); // 进房时补发的进行中测  
   验  
4. quiz.observableOfQuizEnd.subscribe((e:  
    QuizEvent) => { ... }); // 老师结束测验  
5. quiz.observableOfQuizSolution.subscribe((e:  
    QuizEvent) => { ... }); // 老师公布答案  
6. quiz.sendSubmit(json);  
   // 提交答案
```

`QuizEvent` 主要字段：`quizId` / `forceJoin`（是否强制答题，为 `true` 时不应提供关闭入口）/ `endFlag`（已结束）/ `hasSolution`（已提交过）。

说明：UI SDK 已提供完整的测验窗口。仅集成 Core SDK 并需要自建测验界面时，请联系技术支持获取测验页面的接入说明。

11.13 学情报告与课后评价

1. `const info: LPStudyReportInfo = await room.getBlackboardImgList(reportNumber);` // 学情报告中的板书图片
2. `room.isShowEvaluation();` // 下课后是否需要展示课后评价

`isShowEvaluation()` 只反映课程与机构的配置，是否对当前角色展示由应用决定（通常仅对学生展示）。

12. 商品与点赞（LiveShowVM）

大班课中支持商品展示、讲解与直播点赞：

1. `const vm = room.liveShowVM;`
2.
3. `vm.observableOfSellProducts.subscribe((list: LiveProductModel[]) => { ... });` // 商品列表
4. `vm.observableOfProductExplain.subscribe((p: LiveProductModel) => { ... });` // 正在讲解的商品
5. `vm.observableOfSellProductInventoryChange.subscribe(LiveProductInventoryModel) => { ... });`

```

6. vm.observableOfFlashSaleCountChange.subscribe((
    FlashSaleInfo) => { ... });
7. vm.requestNextPage();
   vm.refreshProductList();
8. const r: LiveProductModel[] = await
   vm.requestSearchResults(query);
9. vm.isProductVisible();
   vm.isProductPriceVisible();
10.
11. // 直播点赞
12. vm.requestSendLiveLike(n);
13. vm.getLiveLikeCount();
14. vm.observableOfLiveLikeCountChange.subscribe((n
    number) => { ... });

```

LiveProductModel 主要字段: id / name /
 imgUrl / price / discountPrice / buyUrl /
 desc / isOnShelf / flashSaleInfo 。

相关开关: room.isSellGoodsInLargeClassEnabled() /
 room.isLiveLikeEnabled() 。

12.1 活动推送（学生端）

```

1. vm.observableOfActivePublish.subscribe((a:
   ActivityInfo) => {
2.   if (a.status === 0) { /* 活动已取消，收起卡片 */
     return; }
3.   // 展示活动卡片；isKeepShow 为 false 时建议 5 秒
     后自动收起
4. });

```

13. 教室状态与管控

13.1 上课与下课

```
1. room.isClassStarted();
2. room.requestClassStart(); // 老师上课
3. room.requestClassEnd(); // 老师下课
4. room.observableOfClassStart.subscribe((started:
    boolean) => { ... }); // false 表示已下课
```

13.2 公告

```
1. room.getAnnouncement(); //
    NoticeInfo: content / link
2. room.changeRoomAnnouncement(content, link);
    // 老师修改公告
3. room.requestAnnouncement(); // 主动
    拉取公告
4. room.observableOfAnnouncementChange.subscribe(
    NoticeInfo) => { ... };
```

13.3 全员管控

```
1. room.requestForbidAllAudio(forbid); // 全体
    静音
2. room.observableOfForbidAllAudioStatus.subscribe((
    boolean) => { ... });
3.
4. room.requestForbidRaiseHand(forbid); // 禁
    止举手
5. room.observableOfForbidRaiseHand.subscribe((on:
    boolean) => { ... });
6.
7. room.requestForbidStudentSwitchPPT(forbid); //
    禁止学生翻页
```

```
8. room.getStudentSwitchPPTState(); // 学
   生当前是否可以翻页
```

注意: `requestForbidStudentSwitchPPT(forbid)` 的参数表示「是否禁止」, 而 `getStudentSwitchPPTState()` 返回「是否允许」, 两者含义相反。以上管控接口仅老师 / 助教可调用, 其他角色调用会回调 `LPERrorCode.PermissionDeny`。

13.4 课件与视频主次切换

```
1. room.requestPPTVideoSwitch(isVideoInMain); //
   全员同步切换
2. room.switchPPTVideoLocal(isVideoInMain); // 仅
   本端切换
3. room.isVideoInMain();
4. room.observableOfPPTVideoSwitch.subscribe((video
   boolean) => { ... });
```

13.5 自定义广播

用于在房间内发送 SDK 未封装的自定义业务消息, 例如自定义弹窗、跨端同步业务状态等:

```
1. import { BroadcastEvent } from
   '@baijiayun/livecore';
2.
3. // 发送: 第三个参数表示是否由服务端缓存, 后进入的用户可补收
4. room.sendBroadcast('my_biz_key', { 'foo': 1,
   'bar': 'x' }, true);
5.
6. // 接收
```

```
7. room.observableOfBroadcast.subscribe((e:
BroadcastEvent) => {
8.   if (e.key !== 'my_biz_key') { return; } // 务必按
key 过滤
9.   const v: Record<string, Object> = e.value;
10. });
11.
12. // 进房后补收缓存
13. room.requestBroadcastCache('my_biz_key');
14. room.observableOfBroadcastCache.subscribe((e:
BroadcastEvent) => { ... });
```

使用规则：

- 1. 接收时务必判断 `key`，该事件流中也包含教室内部使用的广播；
- 2. 希望后进入的用户也能收到时，发送时第三个参数传 `true`，并在进房后调用 `requestBroadcastCache(key)` 补收；
- 3. 发送频率不超过 **每秒 5 条、每分钟 60 条**，单条内容序列化后默认不超过 **1024 字符**。

发送失败通过 `observableOfError` 回调：

错误码	说明
<code>InvalidSignalKey</code> (-1020)	<code>key</code> 为空或不在机构允许
<code>InvalidSignalValue</code> (-1021)	内容超长
<code>SignalFrequencyTooHigh</code> (-1022)	发送过于频繁
<code>InvalidUserRole</code> (-1023)	当前角色无权发送

14. 录制与转码

```

1. room.canCloudRecord();           // 当前是否可以开启云端录制
2. room.requestCloudRecord(status); // 开始 / 暂停 / 停止, status 见 CloudRecordStatus
3. room.getCloudRecordStatus();
4. room.observableOfCloudRecordStatus.subscribe((status, number) => { ... });
5.
6. room.requestCloudRecordStartProcessing(); // 生成回放（一般在下课时调用）
7. const st: PlaybackProcessStatus = await room.requestPlaybackStatus();

```

CloudRecordStatus : Stopped (0) / Recording (1) / Paused (2)。

PlaybackProcessStatus.status : 0 未开启云端录制 / 1 已录制未生成回放 / 2 已开始生成回放。

15. LiveRoom 其他 API

```

1. const me: CurrentUser = room.getCurrentUser();
   // userId / userNumber / userName / userType
2. room.getRoomId();
3. room.getRoomInfo(); // RoomInfo: roomId / title / roomType
4. room.isTeacher();
   room.isTeacherOrAssistant();
5. room.getTeacherUser(); // 教室中的老师
6. room.getPresenterUser(); // 当前主讲人
7. room.getWaterMark(); // 水印: url / pos
8. room.getPartnerConfig(); // 机构配置原始对象

```

SDK 还提供一组常用机构配置的读取接口，例如 `isQaEnabled()`、`isHomeworkEnabled()`、`canSendChatImage()`、`isShareEnabled()` / `getShareLink()`、`getAutoOpenCameraStatus()` 等，建议优先使用这些接口读取配置。

举报：

```
1. await room.report(reasons, message); //
   message: ReportedMessage | null
```

16. 后台运行

App 退到后台后，若希望直播声音继续播放、音视频不中断，需要应用自行完成以下三项，缺一不可：

项	作用
① 在 <code>module.json5</code> 中声明后台模式与权限	允许申请长时任务
② 进入教室时申请长时任务	退到后台后应用不被冻结
③ 进入教室时创建并激活媒体会话（AVSession）	系统识别应用正在播放媒体

长时任务与媒体会话属于应用级能力（需指向应用自己的 Ability），因此由应用实现，SDK 不代为处理。

16.1 声明后台模式

```
1. {
2.   "module": {
```

```

3.   "abilities": [
4.     {
5.       // ...
6.       "backgroundModes": ["audioRecording",
7.         "audioPlayback", "wifiInteraction"]
8.     },
9.     "requestPermissions": [
10.      { "name":
11.        "ohos.permission.KEEP_BACKGROUND_RUNNING",
12.        "usedScene": { "abilities": ["EntryAbility"] } }
13.      // ...其余权限见 §2.3
14.    ]
15.  }
16. }

```

16.2 申请长时任务

```

1. import wantAgent from
   '@ohos.app.ability.wantAgent';
2. import { backgroundTaskManager } from
   '@kit.BackgroundTasksKit';
3. import common from
   '@ohos.app.ability.common';
4. import { BusinessError } from '@ohos.base';
5.
6. export class BackgroundTask {
7.   private static running: boolean = false;
8.
9.   static start(context: common.UIAbilityContext):
10.    void {
11.      if (BackgroundTask.running) { return; }
12.      BackgroundTask.running = true;
13.      const info: wantAgent.WantAgentInfo = {

```

```

13.     wants: [{ bundleName:
        context.abilityInfo.bundleName, abilityName:
        context.abilityInfo.name }],
14.     operationType:
        wantAgent.OperationType.START_ABILITY,
15.     requestCode: 0,
16.     wantAgentFlags:
        [wantAgent.WantAgentFlags.UPDATE_PRESENT_FLA
17.     ];
18.     wantAgent.getWantAgent(info).then((agent:
        Object) => {
19.
        backgroundTaskManager.startBackgroundRunning(
20.         context,
        backgroundTaskManager.BackgroundMode.AUDIO_F
        agent)
21.         .catch((e: BusinessError) => {
            BackgroundTask.running = false; });
22.     }).catch((e: BusinessError) => {
        BackgroundTask.running = false; });
23.     }
24.
25.     static stop(context: common.UIAbilityContext):
        void {
26.         if (!BackgroundTask.running) { return; }
27.         BackgroundTask.running = false;
28.
        backgroundTaskManager.stopBackgroundRunning(c
        BusinessError) => {});
29.     }
30. }

```

16.3 创建并激活媒体会话

```
1. import { avSession } from '@kit.AVSessionKit';
2. import common from
   '@ohos.app.ability.common';
3. import { BusinessError } from '@ohos.base';
4.
5. export class AvSessionKeeper {
6.   private static session: avSession.AVSession |
     null = null;
7.   private static starting: boolean = false;
8.
9.   static async start(context:
     common.UIAbilityContext, roomTitle: string):
     Promise<void> {
10.    if (AvSessionKeeper.session !== null ||
        AvSessionKeeper.starting) { return; }
11.    AvSessionKeeper.starting = true;
12.    try {
13.      const session = await
        avSession.createAVSession(context, 'LiveRoom',
        'audio'); // 类型使用 'audio'
14.      AvSessionKeeper.session = session;
15.      await session.setAVMetadata({ assetId: 'live-
        room', title: roomTitle });
16.      await session.setAVPlaybackState({ state:
        avSession.PlaybackState.PLAYBACK_STATE_PLAY
        });
17.      await session.activate(); // 必须激活
18.    } catch (e) {
19.      AvSessionKeeper.session = null;
20.    }
21.    AvSessionKeeper.starting = false;
22.  }
23.
24.  static stop(): void {
25.    const session = AvSessionKeeper.session;
```

```
26.     if (session === null) { return; }
27.     AvSessionKeeper.session = null;
28.     session.destroy().catch((e: BusinessError) =>
29.         {});
30. }
```

16.4 调用时机

在教室页面的生命周期中成对调用，不要放在 Ability 的

`onCreate` / `onDestroy` 中：

```
1. // 进入教室
2. BackgroundTask.start(getContext(this) as
   common.UIAbilityContext);
3. AvSessionKeeper.start(getContext(this) as
   common.UIAbilityContext, roomTitle);
4.
5. // 退出教室
6. BackgroundTask.stop(getContext(this) as
   common.UIAbilityContext);
7. AvSessionKeeper.stop();
```

说明：申请失败时不要中断进房流程。媒体会话激活后，系统的播控中心与锁屏界面会显示一张媒体卡片，属于系统行为。

SDK 提供「退到后台后是否继续播放声音」的偏好开关，可接入应用的设置项：

```
1. room.setAudioBackOpen(false); // 默认 true
2. room.isAudioBackOpen();
```

该开关只决定是否在后台继续播放，前提仍是完成以上三项。

17. 枚举 / 接口说明

17.1 LPEmitter

SDK 中所有 `observableOfXxx` 事件均为

`LPEmitter<T>` 类型：

```
1. class LPEmitter<T> {  
2.   subscribe(observer: LPObserver<T>):  
    LPDisposable;  
3.   unsubscribe(observer: LPObserver<T>): void;  
4.   getValue(): T | null;  
5. }  
6. type LPObserver<T> = (value: T) => void;  
7. interface LPDisposable { dispose(): void;  
    isDisposed(): boolean; }
```

- **状态类事件带粘性**：新订阅者会立即收到最近一次的值，因此组件晚于教室创建时也能拿到当前状态；进入 / 离开等一次性事件不带粘性。
- `subscribe()` 返回 `LPDisposable`，在组件销毁时调用 `dispose()` 即可反订阅：

```
1. private sub: LPDisposable | null = null;  
2. aboutToAppear(): void { this.sub =  
    room.observableOfClassStart.subscribe((v) => {  
      ... }); }  
3. aboutToDisappear(): void { this.sub?.dispose(); }
```

一问一答类的接口（如上传、红包、查询回放状态等）以

`Promise` 返回结果。其余接口返回 `void`，执行失败时通过 `room.observableOfError` 回调。

17.2 常用枚举

```
1. enum LPUserType { Student = 0, Teacher = 1,
    Assistant = 2, Visitor = 3 }
```

本 SDK 当前支持大班课房型（ `LPRoomType.Multi` ）。

18. 错误码

错误通过 `LPRoomError` 回调： `code: LPErrorCode` / `message: string` 。

枚举	值	说明
<code>Unknown</code>	-1	未知错误
<code>NetworkFailure</code>	-1001	网络请求失败
<code>NetworkMobile</code>	-1002	当前为移动网
<code>InvalidParams</code>	-1006	参数错误
<code>OpenAudioCameraFailed</code>	-1009	打开摄像头 / 麦克风失败
<code>RoomServerLoseConnection</code>	-1011	与教室服务断（SDK 会自动重连）
<code>LoginConflict</code>	-1013	账号在其他端
<code>PermissionDeny</code>	-1014	无操作权限
<code>MediaPlayFailed</code>	-1016	播放失败
<code>MessageSendForbid</code>	-1017	已被禁言
<code>InvalidSignalKey</code>	-1020	自定义广播 ke

		不合法
InvalidSignalValue	-1021	自定义广播内
SignalFrequencyTooHigh	-1022	发送过于频繁
InvalidUserRole	-1023	当前角色无权
PlayerLag	-1025	播放卡顿
MediaSubscribeTimeout	-1026	拉流超时

音视频进房失败：message 为 ERR_AV_ENTER_ROOM_FAILED 时，建议提示用户重试，并调用 room.retryEnterAvRoom() （返回 false 表示已无法继续重试）。

登录失败：message 可能为以下常量：

常量	说明
ERR_LOGIN_UNIQUE_CONFLICT	账号已在其他端登录 enterRoomIgnoreT 强制进入)
ERR_LOGIN_KICK_OUT	已被请出教室
ERR_LOGIN_MAX_STUDENT	教室人数已满
ERR_LOGIN_AUDITION_END	试听已结束
ERR_LOGIN_CLASS_EXPIRED	课程已过期

19. 版本说明

只列出影响集成的变化，完整更新日志见包内 CHANGELOG.md 。

1.9.1

修复部分机型（如折叠屏）前置摄像头无画面的问题，以及折叠屏开合后摄像头未切换到可用镜头的问题。无接口变更，从 1.9.0 升级无需修改代码。

1.9.0

新增学情报告与课后评价相关接口：

`LiveRoom.getBlackboardImgList()`、`LiveRoom.isShowEvaluation()`、`parseStudyReportInfo()`。无破坏性变更。

1.8.0

`LiveRoom.playAVClose(userId)` 语义调整：由「只停画面、保留声音」调整为「音视频全部停止」。原有「只停画面」的用法请改为新增的 `playAudio(userId)`：

1. - `room.playAVClose(userId)`
2. + `room.playAudio(userId)`

新增 `getRemotePlayMode(userId)` 查询当前观看模式。

`DocListVM.hasPPTAuth()` 语义调整：由「老师是否禁止学生翻页」调整为「自己是否被单独授予课件操作权限」。原用于判断能否翻页的代码请改为订阅

`observableOfPageFlipAuth`：

1. - `if (docListVM.hasPPTAuth()) { ... }`
2. +
`docListVM.observableOfPageFlipAuth.subscribe((canFlip: boolean) => { ... })`

新增：学生课件操作授权（`observableOfStudentPPTAuth` /

requestStudentPPTAuthChange)、muteRemoteAudio / muteRemoteVideo、requestAnnouncement。

部分接口补充了调用前置校验：被禁言时发送消息、非老师 / 助教调用管控接口等，现在会通过 observableOfError 返回错误。

1.7.0

以下接口更名，直接使用 WhiteboardPanel 且未传入这些回调的集成方无需修改：

原名称	新名称
WhiteboardPanel.onAnimPptError	onPPTError
WhiteboardPanel.onAnimPptLoadFinish	onAnimPPTLoadFinish
DocListVM.hasH5Doc()	isContained

setDisableOverTeacherMaxPage 默认值调整为 true（学生不能翻过老师已翻到的页）。需要允许学生自由翻页时，请显式调用 setDisableOverTeacherMaxPage(false)。

新增视频首帧事件 LPPlayer.observableOfReadyToPlay / LPRecorder.observableOfReadyToPlay。

1.5.0

支持同一用户同时发布多路流（如摄像头 + 屏幕共享）。PublishedMedia / RemoteSlotInfo 新增 mediaId、sourceType 等字段，getActiveUserList()、getRemoteLayout() 等接口可能返回 userId 相同的多条记录。升级时请检查：

- 以 userId 作为 Map / Set 键缓存这些记录的代码，改为以 mediaId 作为键；

2. 统计台上人数的代码，先过滤 `sourceType ===`
`MediaSourceType.MainCamera` ；
3. 查询某人摄像头状态的代码，补充 `sourceType` 条件。

建议升级后验证「老师开启屏幕共享时，学生端的台上人数、名单与窗位显示是否正常」。

20. 常见问题

1. 学生端的翻页、画笔比老师端晚一两秒？

属于正常现象。台下学生观看的合流画面存在一定延迟，SDK 会将翻页、画笔等教室消息延后相应时间处理，使课件与画面保持同步。延迟时长由机构配置决定。

2. 退到后台后没有声音？

请检查是否完成了 §16 的三项配置，尤其是媒体会话是否已调用 `activate()` 。

3. 如何反馈问题？

请将 `LPFileLog.getDir()` 目录整体打包，连同问题现象、发生时间、房间号一并提供给技术支持。