Android 直播 Core SDK

- **Gitlab链接**
- **JavaDoc链接**
- Platform: API 21+
- CPU: ARM-v7a, ARM64-v8a
- IDE: **Android Studio** Recommend
- Change Log
- 3.X SDK 文档
- 3.X 升级 4.0 SDK API 变更说明

功能介绍

百家云直播 Android SDK提供了 Live Core SDK、开源大班课横版三分屏 UI SDK、开源大班课竖版企业模板 UI SDK、开源大班课直播带货 UI SDK 、开源大班 TV UI SDK 和 开源专业小班课UI SDK。

- UI 库基于 Core 实现，提供了一个针对教育场景下师生互动模板，主要包括师生一对一音视频互动，多人音视频互动，课件展示、文字聊天等功能，可以快速接入，集成工作量小，适合需要快速上线的同学，该库开源。
- Core 为核心库，涵盖了直播间几乎所有的功能，包括音视频推拉流、信令服务器通信、聊天服务器通信、课件展示与画笔绘制等功能。

1. 概念

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

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

	显示老师的视频画面
自习室	在线连麦自习、辅导

2. 引入 SDK

2.1. 添加 maven 仓库

从 4.0 SDK 起，引入了新的 `nexus.baijiayun.com` 仓库。

gradle 7.1 及以上

```
maven {
        allowInsecureProtocol true
        url 'http://nexus.baijiayun.com/nexus/content/groups/apublic/'
    }
maven { url 'https://git2.baijiashilian.com/open-android/maven/raw/master/' }
```

gradle 7.1 以下

```
maven {url 'http://nexus.baijiayun.com/nexus/content/groups/apublic/'}
maven { url 'https://git2.baijiashilian.com/open-android/maven/raw/master/' }
```

2.2. 添加依赖

最新版本请点击自取，Releases · Open Android / BJLiveCore-Android · GitLab

```
dependencies {
    implementation 'com.baijiayun.live:liveplayer-sdk-core:4.2.0'
}
```

2.3. 版本号说明

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

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

首次集成建议选择最新正式版本(版本号中不带有 `alpha` 、 `beta` 字样)，**版本升级后请仔细阅读**

2.4. 设置专属域名

专属域名（个性域名）从百家云账号中心获取，需要在进入直播点播和回放之前设置。例如专属域名为 `demo123.at.baijiayun.com` ，则前缀为 `demo123` ，参考 专属域名说明。

```
LiveSDK.customEnvironmentPrefix = "demo123";
```

2.5. 初始化 SDK

`LiveSDK.init(context)` 推荐在同意**隐私政策**之后调用。

```
LiveSDK.init(applicationContext);
```

注： sdk 内部默认会初始化腾讯 X5 内核，请在同意隐私政策之后初始化。

另外，LiveSDK 对外暴露了以下静态变量以自定义配置：

```
    /**
     * 后台音频开关
     */
    public static boolean isAudioBackOpen = true;

    /**
     * 静态课件默认压缩参数，默认值为1080，值越小图片压缩越厉害，取值范围为 [1,16384]
     */
    public static int STATIC_PPT_SIZE = 1080;

    /**
     * 静态课件是否按原图尺寸加载（默认false），设置为 true STATIC_PPT_SIZE 不再起作用
     */
    public static boolean SHOW_STATIC_PPT_ORIGIN_SIZE;

    /**
     * 跑马灯文案，默认走后台配置，非必要不建议通过这里写死
     */
    public static String LIVE_HORSE_LAMP;
```

3. 教室管理

3.1. 要点说明

- 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，需要调用 `LiveSDK.enterRoom` ，如果成功会回调到 `LPLaunchListener.onLaunchSuccess(LiveRoom liveRoom)` ，返回的LiveRoom实例可以获取到各个功能对应的 **ViewModel**，后文会对进入教室和各个 ViewModel 具体说明。

- RxJava订阅之后在不使用时需要反订阅，例如

```
// 监听上课
Disposiable disposiable = liveRoom.getObservableOfClassStart().subscribe(co
// 在onDestroy时，需要反订阅
disposiable.dispose();
```

本文为了简单起见，忽略了反订阅操作。

3.2. 进入直播间

LiveSDK.enterRoom 目前提供房间号、参加码和 url 三种方式进入房间，接口如下

```
/**
 * @param context
 * @param signEnterRoomModel 签名进房间
 * @param listener   进房间回调
 */
public static LiveRoom enterRoom(@NonNull final Context context, LPSignEnterRoomModel signEnterRoomModel, LPLaunchListener launchListener)

```

```
/**
 * @param context
 * @param joinCodeEnterRoomModel 参加码进房间
 * @param listener 进房间回调
 */
public static LiveRoom enterRoom(@NonNull final Context context, LPJoinCodeEnterRoomModel joinCodeEnterRoomModel, LPLaunchListener launchListener)

/**
* @param context
* @param url 分享的链接或者PC进教室链接
* @param listener  进房间回调
*/
public static LiveRoom enterRoom(Context context, String url, final LPLaunchListener listener)
```

LPJoinCodeEnterRoomModel 字段说明

```
/**
 * 参加码
 */
public String joinCode;

/**
 * 用户昵称
 */
public String userName;

/**
 * 用户头像
 */
public String userAvatar;
```

```
15.
16. /**
17.  * 用户角色
18.  */
19. public LPConstants.LPUserType userType;
20.
21. /**
22.  * 自定义字段，透传给直播服务器
23.  */
24. public String customStr;
```

LPSignEnterRoomModel 字段说明

```
1. /**
2.  * 房间号
3.  */
4. public long roomId;
5.
6. /**
7.  * 用户名称
8.  */
9. public String userName;
10.
11. /**
12.  * 用户唯一编号
13.  */
14. public String userNumber;
15.
16. /**
17.  * 用户头像
18.  */
19. public String userAvatar;
20.
21. /**
22.  * 分组id
```

```
 */
public int groupId;

/**
 * 用户类型
 */
public LPConstants.LPUserType userType;

/**
 * 签名
 * 请求接口参数签名, 签名由 (roomId, groupId, userNumber, userName, userType, userAvatar) 6个参数生成
 * refer : http://dev.baijiayun.com/default/wiki/detail/11#h0-4
 */
public String sign;

/**
 * 自定义字段，透传给直播服务器
 */
public String customStr;

/**
 * 被替换用户userNumber（仅云摄功能需要设置）
 */
public String replaceUserNumber;

/**
 * 被替换用户的角色（仅云摄功能需要设置）
 */
public String replaceUserRole;
```

3.3. 进入房间回调说明

```java
public interface LPRoomStatusListener {
    /**
     * 进房间进度回调，step/totalStep 即进教室百分比
     * @param step 当前步骤
     * @param totalStep 总步骤数
     */
    void onLaunchSteps(int step, int totalStep);

    /**
     * 进房间出错回调
     * @param error
     */
    void onLaunchError(LPError error);

    /**
     * 进房间成功回调
     * @param liveRoom
     */
    void onLaunchSuccess(LiveRoom liveRoom);

    /**
     * 直播过程中的错误回调
     *
     * @param error
     */
    void onLivingError(LPError error);

    /**
     * 退出房间回调
     */
    void onQuitRoom();
}
```

注： 与服务器断开连接会报 `lpError.code = CODE_ERROR_ROOMSERVER_LOSE_CONNECTION` 的错

误，这个时候可以检测网络状态，如果有网可以进行断网重连。一般的重连步骤为quitRoom（退出教室），然后重新enterRoom（进入教室）就行了，LiveRoom 为新实例，UI资源也需要适时销毁和重新创建。

3.4. 离开直播间

一般在 **Activity**的 `onDestroy()` 中调用

```
liveRoom.quitRoom();
```

3.5. getViewModel()

`LiveRoom` 是整个 SDK 的上下文，提供以下一系列 **getVM** 函数。每个 **ViewModel** 代表一个功能集，会在下边的章节展开介绍。

```
/**
    * 获取推流管理模块，包括开关摄像头、麦克风、切换推流清晰度等操作
    */
   LPRecorder getRecorder();

   /**
    * 获取拉流管理模块，包括拉指定用户视频、音频、停止拉流等操作
    */
   LPPlayer getPlayer();

   /**
    * 获取发言列表模块
    */
   SpeakQueueVM getSpeakQueueVM();

```

```
16.     /**
17.      * 获取媒体模块
18.      * @return
19.      */
20.     LPMediaVM getMediaVM();
21.
22.     /**
23.      * 获取聊天模块
24.      */
25.     ChatVM getChatVM();
26.
27.     /**
28.      * 获取文档管理模块
29.      */
30.     DocListVM getDocListVM();
31.
32.     /**
33.      * 画笔
34.      * @param uiInterface
35.      * @return
36.      */
37.     ShapeVM
    newShapeVM(ShapeVM.LPShapeReceiverListener
    uiInterface);
38.
39.     /**
40.      * 云盘文件列表
41.      */
42.     CloudFileVM getCloudFileVM();
43.
44.     /**
45.      * 在线用户模块
46.      */
47.     OnlineUserVM getOnlineUserVM();
48.
49.     /**
```

```
     * 测验模块
     */
    QuizVM getQuizVM();

    /**
     * 工具箱 ViewModel
     */
    ToolBoxVM getToolBoxVM();

    /**
     * 直播带货 ViewModel
     */
    LiveShowVM getLiveShowVM();

    /**
     * 自习室VM
     */
    StudyRoomVM getStudyRoomVM();

    /**
     * 企业直播 ViewModel
     * @return
     */
    LiveEEVM getLiveEEVM();
```

4. 本地推流 (LPRecorder)

发布音视频主要使用到了 **LPRecorder**。

```
// 进房间成功后通过 liveRoom.getRecoder() 拿到
LPRecorder recorder = liveRoom.getRecorder();
```

4.1. 基础功能

```
/**
 * 设置本地摄像头预览，默认从前置摄像头采集
 *
 * @param cameraView
 */
void setPreview(LPCameraView cameraView);

/**
 * 设置本地摄像头预览
 * @param isFrontCamera 是否是前置摄像头
 * @param cameraView
 */
void setPreview(boolean isFrontCamera,
LPCameraView cameraView);

/**
 * 停止预览
 */
void stopLocalPreview();

/**
 * 获取上行视频采集的LPCameraView
 *
 * @return
 */
LPCameraView getPreview();

// 发布流
recorder.publish();
// 打开音频
recorder.attachAudio();
// 打开视频
recorder.attachVideo();
// 同时打开音频和视频
recorder.attachAVideo();
```

```
// 关闭音频
recorder.detachAudio();
// 关闭视频
recorder.detachVideo();
// 同时关闭音频和视频
recoder.detachAVideo();
// 是否正在推视频
boolean isVideoAttached();
// 是否正在推音频
boolean isAudioAttached();
// 停止发布流
recorder.stopPublishing();
// 流是否正在上传
boolean isPublishing();

/**
 * 摄像头是否打开回调
 *
 * @return
 */
Flowable<Boolean>
getObservableOfCameraOn();

/**
 * 麦克风是否打开回调
 *
 * @return
 */
Flowable<Boolean> getObservableOfMicOn();
```

例如:发布本地音频时，先调用 `recorder.publish();` 然后再调用 `recorder.attachAudio();` 即可。
注意 发布视频时需要先设置本地视频采集的preview，然后再调用 `recorder.attachVideo();`

```
// 也可在布局文件里创建
LPCameraView cameraView = new LPCameraView(context);
recorder.setPreview(cameraView);
```

LPCameraView为本地视频预览，提供了 SurfaceView 和 TextureView 两种 View 进行视频渲染（如果为 BRTC 底层，目前仅支持SurfaceView）

```
cameraView.setViewType(LPPConstants.LPVideoVie
// 或者
cameraView.setViewType(LPConstants.LPVideoView
// 获得当前view类型
LPConstants.LPVideoViewType cameraView.getViewType();
```

注意： 请确保在采集前 APP 已经获得相应麦克风或者摄像头的权限，`recorder.setPreview(cameraView)` 建议在 `recorder.publish();` 之前调用。

设置采集分辨率如下，LPResolutionType 说明见 此处

```
// 设置采集分辨率
void setCaptureVideoDefinition(LPConstants.LPResolutio definition);

/**
 * 获取当前分辨率
 *
 * @return
 */
LPConstants.LPResolutionType getVideoDefinition();
```

4.2. 推流镜像

镜像分本地预览镜像（仅影响预览的镜像）和推流镜像（决定推出去的流的镜像），BJYRtcCommon.VideoMirrorMode 说明见 此处

```
/**
 * 设置本地预览是否镜像
 * @param
 */
void
setLocalVideoMirrorMode(BJYRtcCommon.VideoMirr
 mode);

/**
 * 获取本地预览镜像模式
 * @return
 */
BJYRtcCommon.VideoMirrorMode
getLocalVideoMirrorMode();

/**
 * 获取推流镜像模式
 * @return
 */
BJYRtcCommon.VideoMirrorMode
getEncodeVideoMirrorMode();

/**
 * 设置推流镜像
 * @param mode
 */
void
setEncVideoMirrorMode(BJYRtcCommon.VideoMirror
 mode);
```

4.3. 基础美颜

美颜功能分磨皮和美白两部分，支持单独设置，范围均为[0.0,1.0]。

```
/**
 * 设置磨皮级别
 * @param level [0.0,1.0]
 */
void setBeautyLevel(float level);

/**
 * 获取磨皮级别
 * @return level [0.0,1.0]
 */
float getBeautyLevel();

/**
 * 设置美白级别
 * @param level [0.0,1.0]
 */
void setWhitenessLevel(float level);

/**
 * 获取美白级别
 * @return level [0.0,1.0]
 */
float getWhitenessLevel();
```

4.4. 屏幕分享

```
/**
 * 停止屏幕分享
 */
```

```
void stopScreenCapture();

/**
 * 暂停屏幕分享
 */
void pauseScreenCapture();

/**
 * 恢复屏幕分享
 */
void resumeScreenCapture();

/**
 * 屏幕分享状态回调
 */
Observable<LPConstants.LPScreenShareState>
getObservableOfScreenShareState();

/**
 * 是否正在屏幕分享
 * @return
 */
boolean isScreenSharing();
```

4.5. 课前预览

课前预览可实现未上课状态下打开摄像头并且采集画面。

```
/**
 * 设置本地摄像头预览（实现未上课不推流的预览）
 * @param cameraView
 */
void startLocalPreview(LPCameraView
cameraView);

```

```
/**
 * 设置本地摄像头预览（实现未上课不推流的预览）
 * @param isFrontCamera 是否是前置摄像头
 * @param cameraView
 */
void startLocalPreview(boolean isFrontCamera,
LPCameraView cameraView);
```

4.6. PCM

PCM 为麦克风原始采集数据，可用于音频转字幕等处理。

注： 3.19.2 版本新增 pcm 音频数据回调接口，**仅 BRTC 底层支持**

BRTCListener.BRTCAudioFrameListener 说明见 此处

```
/**
 * 设置PCM音频数据回调
 *
 * @param audioFrameListener
 */
void
setAudioFrameListener(BRTCListener.BRTCAudioFra
 audioFrameListener);
```

4.7. 其它

首帧回调，可用于优化打开摄像头的体验。attachVideo() 打开设置头后立即显示 loading 占位图，在 `onReadyToPlay()` 回调里主动隐藏 loading。

```
/**
 * 添加首帧回调
 */
```

```
void addRecorderListener(LPRecorderListener listener);

/**
 * 移除首帧回调
 */
void removeRecorderListener(LPRecorderListener listener);
```

双路流，同时推一大一小两路视频流。需要后台开启 `enable_use_dual_stream` 配置项，并且配置的小流分辨率 `video_stream_low_height` 需要小于大流的分辨率。

```
/**
 * 开/关视频双流模式
 * @param enable
 */
void enableDualStreamMode(boolean enable);
```

音乐模式，设置音乐模式后音频码率会变大，适用于想获得更好音频收听体验的场景。

```
/**
 * 设置音乐模式
 * @param isOpen
 */
void changeMusicModeOn(boolean isOpen);

/**
 * 获取音乐模式状态
 * @return
 */
boolean isMusicModeOn();
```

其它不太常用 API 如下：

```
// 切换摄像头(如果有)
void switchCamera();
// 获得系统摄像头数量
int getCameraCount();

/**
 * 上行丢包率
 *
 * @return
 */
Flowable<BJYRtcEventObserver.LocalStreamStats>
 getObservableOfUpPacketLossRate();

/**
 * 摄像头方向，默认true为前置，false为后置
 */
boolean getCameraOrientation();

/**
 * 视频截图
 *
 */
void takeVideoScreenshot();

/**
 * 视频截图
 *
 * @return 截图的绝对路径
 */
Flowable<LPVideoScreenshot>
getObservableOfVideoScreenshot();
```

5. 音视频拉流 (LPPlayer)

5.1. 发言列表

用户（老师/学生）进入教室之后，如果教室内已经有人上麦发言，我们称之为 ActiveUser，可以通过如下方法监听获取到 ActiveUser。

```
Observable<Lis<IMediaModel>> obs = liveRoom.getSpeakQueueVM().getObservableOfActi
```

至此，已经获取到了教室里所有上麦的用户及其音视频的状态，如果之后有新的用户上麦发言或者 ActiveUser 中某个用户的音视频状态发生了变化，比如老师关闭了麦克风或者摄像头等，会回调 `getObservableOfMediaPublish()` ，例如

```
liveRoom.getSpeakQueueVM().getObservableOfMed
.subscribe(new Consumer<IMediaModel>() {
    @Override
    public void accept(IMediaModel iMediaModel) {
        // 拉流
        //
player.playVideo(iMediaModel.getMediaId());
    }
});
```

其中，IMediaModel包含用户信息、音视频状态等。

```
public interface IMediaModel {
    String getMediaId(); //获取流唯一标识
    boolean isVideoOn(); //是否有视频
    boolean isAudioOn(); //是否有音频
    IUserModel getUser(); //获取对应的User
    boolean isMixedStream(); // 是否是合流
```

```
    List<LPConstants.VideoDefinition>
getVideoDefinitions(); //获取支持的清晰度列表
    //等等...
}
```

当获得这个用户对象的 `mediaId` 就能开始拉流，播放其音视频了。

5.2. LPPlayer 拉流播放器

```
LPPlayer player = liveRoom.getPlayer();
String mediaId = iMediaModel.getMediaId();
// 播放音频
player.playAudio(mediaId);
// 播放音视频
player.playVideo(mediaId, videoView);
// 只拉视频，仅对合流生效。
// RTC 流：playVideo(String mediaId, LPVideoView
videoView) + muteRemoteVideo(String mediaId,
true) 实现
player.playVideo(mediaId, videoView, true);
// 关闭音视频流
player.playAVClose(mediaId);
```

注意： `player.playVideo(mediaId, videoView)` 中的 videoView 为显示视频的 view

```
// 也可在布局文件里创建
LPVideoView videoView = new
LPVideoView(context);
```

```
// 增加播放音视频回调
void addPlayerListener(LPPlayerListener listener);
// 移除播放音视频回调
```

```
void removePlayerListener(LPPlayerListener listener);
// 是否在拉指定用户的音频流
boolean isAudioPlaying(String mediaId);
// 是否在拉指定用户的视频流
boolean isVideoPlaying(String mediaId);
// 打开/关闭所有拉流音频，非合流（纯 webrtc）场景有效
void muteAllRemoteAudio(boolean mute);
// 打开/关闭所有拉流视频，非合流（纯 webrtc）场景有效
void muteAllRemoteVideo(boolean mute);
// 打开/关闭指定用户流的视频，非合流（纯 webrtc）场景有效
void muteRemoteVideo(String mediaId, boolean mute);
// 打开/关闭指定用户流的音频，非合流（纯 webrtc）场景有效
void muteRemoteAudio(String mediaId, boolean mute);
// 设置大班课推流直播学生拉流 cdn 清晰度 （只有大班合流和推流直播适用）
void setCDNResolution(LPConstants.LPCDNResolution resolution);
/**
 * 设置拉流模式
 * @param mediaId
 * @param streamType VIDEO_STREAM_HIGH 大流 VIDEO_STREAM_LOW 小流
*/
void setRemoteVideoStreamType(String mediaId, BJYRtcCommon.DualStreamType streamType);
// 离开房间
void leaveRoom();
// 离开房间&销毁资源
```

```
void release();
```

除了 webrtc 流，SDK 还提供拉 rtmp 直播流，如合流、暖场视频、插播视频等场景。 `LPPlayer` 提供如下一系列接口

```
/**
 * 播放指定rtmp流的音视频
 * @param mediaId      流id
 * @param videoView    渲染视图
 */
void playVideoOfRtmpStream(String mediaId, LPVideoView videoView);

/**
 * 播放指定rtmp流的纯音频
 * @param mediaId   流id
 */
void playAudioOfRtmpStream(String mediaId);

/**
 * 暂停指定rtmp流
 * @param mediaId  流id
 */
void pauseRtmpStream(String mediaId);

/**
 * 销毁播放器
 * @param mediaId
 */
void stopRtmpStream(String mediaId);

/**
 * 恢复播放指定rtmp流
 * @param mediaId  流id
 */
```

```
30.    void resumeRtmpStream(String mediaId);
31.
32.    /**
33.     * 设置播放倍数
34.     * @param mediaId   流id
35.     * @param rate      倍数（0.5~2.0）
36.     */
37.    void setRateOfRtmpStream(String mediaId,
       float rate);
38.
39.    /**
40.     * 快进/快退指定rtmp流
41.     * @param mediaId       流id
42.     * @param seekToTime    快进/快退时间戳，单位
       秒
43.     */
44.    void seekRtmpStream(String mediaId, int
       seekToTime);
45.
46.    /**
47.     * 获取指定rtmp流的当前播放时间点
48.     * @param mediaId     流id
49.     * @return            当前播放时间点（单位秒）
50.     */
51.    int getCurrentPositionOfRtmpStream(String
       mediaId);
52.
53.    /**
54.     * 获取指定rtmp流的总时长
55.     * @param mediaId    流id
56.     * @return           总时长（单位秒）
57.     */
58.    int getDurationOfRtmpStream(String mediaId);
59.
60.   /**
61.     * 获取cdn数量
```

```
     * @return
     */
    int getCDNCountOfMixStream();

    /**
     * cdn数量更新
     * @return
     */
    Observable<Integer>
getObservableOfCDNCountOfMixStream();

    /**
     * 手动切换cdn
     * @return true: 切换成功 false: 切换失败
     */
    boolean switchCDNOfMixStream(int index);

    /**
     * 当前cdn index
     * @return
     */
    int getCDNIndexOfMixStream();
```

5.3. 同一用户多路流

老师在推摄像头视频数据的同时，也可以进行屏幕分享、播放媒体文件、推辅助摄像头。

对于大班课班型，默认屏幕分享、媒体文件会自动替换老师视频，即 playVideo 传进来的 view 在播放老师摄像头数据时，以屏幕分享为例，老师进行了屏幕分享，那么这个 view 会立即拉老师的屏幕分享的流，播屏幕分享，老师结束了屏幕分享即切回摄像头播放。一般能满足大部分场景了，无需做任何处理。老师播放媒体文件也是如此。

如果需要同时播放老师的摄像头视频和屏幕分享，需要指定不自动替换老师的流，即

```
// 是否自动播放老师的屏幕分享和媒体文件并替换调老师的视频
LiveSDK.AUTO_PLAY_SHARING_SCREEN_AND_MEDIA = false;
```

需要在enterRoom之前设置。

此时，正在播放老师的视频，还是以屏幕分享为例，老师进行了屏幕分享，不会替换掉老师的视频了，此时会回调 `getObservableOfMediaPublish()` 方法，通知到UI层有新的流到达，并且 user 是老师，这样就可以通过 playVideo 传入新的 view 播放老师的屏幕分享了。并且可以通过 `LPConstants.MediaSourceType getMediaSourceType()` 获取媒体的视频类型。

如果是进教室之前，老师已经在屏幕分享了，那么这个时候进教室请求的ActiveUser中，老师的 `boolean hasExtraStreams();` 就为true了，可以通过 `List<? extends IMediaModel> getExtraStreams();` 获取到屏幕分享流信息。

最后，老师辅助摄像头始终走**不自动替换**的逻辑；PC端作为学生时，某些课也可以进行屏幕分享，学生屏幕分享始终走**自动替换**的逻辑。

6. 发言管理模块（SpeakQueueVM）

举手发言相关 API 都在 `SpeakQueueVM` 中，可以通过 `liveRoom.getSpeakQueueVM()` 获得。

6.1. 举手发言

6.1.1. 学生端

```
// 请求举手
requestSpeakApply();
// 支持举手倒计时回调
requestSpeakApply(OnSpeakApplyCountDownLister
 listener);
// 取消举手
cancelSpeakApply();
// 学生监听老师同意或者拒绝举手
Observable<IMediaControlModel>
getObservableOfSpeakResponse();
// 学生响应是否接受老师邀请发言
void sendSpeakInvite(int confirm);
// 邀请发言监听
Observable<LPSpeakInviteModel>
getObservableOfSpeakInvite();
// 老师远程控制本地音视频监听
Observable<IMediaControlModel>
getObservableOfMediaControl();
/**
    * 老师同意申请的结果返回（上麦人数超限）
    * <p>
    * 在老师使用speak_apply_res同意某人的举手时，
如音视频并发已达上限speak_apply_res将不会被处
理，
    * 无论成功与否此次操作的结果都将通过
speak_apply_res_result告知老师（用于UI上的状态变
化）。
    *
    * @return
    */
Observable<IMediaControlModel>
getObservableOfSpeakApplyResResult();
```

学生请求发言后，会在百家云老师端接受到这个事件，老师可以选择同意或者拒绝，同意后学生即可调用 `LPRecorder` 的推流方法进行推流，当然如果您的 APP 没有举手流程，学生也可以直接上麦推流。

6.2.2. 老师端

```
// 收到学生举手申请回调
Observable<IMediaModel> getObservableOfSpeakApply();
// 同意举手申请
void agreeSpeakApply(String userId);
// 拒绝举手申请
void disagreeSpeakApply(String userId);
// 关闭其他人发言
void closeOtherSpeak(String userId);
/**
    * 控制学生音频或视频
    *
    * @param userId    学生id
    * @param isVideoOn 是否打开视频
    * @param isAudioOn 是否打开音频
    * @return 是否成功
    */
void controlRemoteUser(String userId, boolean isVideoOn, boolean isAudioOn);
// 邀请学生发言
boolean sendSpeakInviteReq(String userId, boolean invite);
// 学生是否接受上麦邀请监听
Observable<LPSpeakInviteConfirmModel> getObservableOfSpeakInviteRes();
```

6.2. 主讲人

老师默认为主讲人角色，直播过程中老师可以设置某个主讲为主讲人。

```
    /**
     * 获取当前主讲人，需要在 active_user_res 回调后获取，否则时机可能太早返回 null
     *
     * @return
     */
    String getPresenter();

    /**
     * 是否是主讲人
     * @param user
     * @return
     */
    boolean isPresenterUser(IUserModel user);

    /**
     * 主讲人切换回调
     *
     * @return Observable<String> 被切换人的 userId
     */
    Flowable<String> getObservableOfPresenterChange();

    /**
     * 主讲人进房间回调
     *
     * @return
     */
    Observable<String> getObservableOfPresenterIn();

```

```
    /**
     * 切换主讲人，仅老师和开启 live_enable_assistant_turn_presenter 配置项的助教支持
     *
     * @param userId 被切换为主讲人的userId
     */
    void requestSwitchPresenter(String userId);
```

6.3. 其它

```
    /**
     * 举手列表
     *
     * @return
     */
    List<IUserModel> getApplyList();

    /**
     * 是否达到最大上麦人数
     *
     * @return
     */
    boolean isSpeakersFull();
```

7. 在线用户（OnlineUserVM）

在线用户模块可以通过 `liveRoom.getOnlineUserVM()` 获得

7.1. 绑定 adapter

以下两个方法，可以方便高效的绑定 UI 的 Adapter。

```
/**
    * 用户总数，adapter绑定
    *
    * @return
    */
   int getUserCount();

   /**
    * 返回指定 position 下的用户，adapter绑定
    *
    * @param position
    * @return
    */
   IUserModel getUser(int position);
```

7.2. 用户进出回调

```
   /**
    * 用户进教室监听，仅当房间人数小于 100 人才回调
    */
   Observable<IUserInModel>
getObservableOfUserIn();

   /**
    * 用户退出监听，仅当房间人数小于 100 人才回调
    *
    * @return
    */
   Observable<IUserModel>
getObservableOfUserOut();
   /**
    * 人员变化，包括虚拟人数，没有 100 人的限制，实
时回调
    */
```

```
    Observable<Integer>
getObservableOfOnLineUserCount();
```

由于服务器压力，房间人数大于100人时，不再广播用户进入和退出，所以提供了一个加载更多用户的接口。（每次加载30个）

```
/**
     * 获取更多在线用户，page_size=30
     * 按用户分组再加上大分组
     */
    void loadMoreUser();

    /**
     * 获取指定分组的更多在线用户，page_size=30
     *
     * @param groupId 分组ID
     */
    void loadMoreUser(int groupId);

    /**
     * loadMoreUser 的响应
     * 更多用户列表
     *
     * @return
     */
    Observable<List<IUserModel>>
getObservableOfUserMore();
```

此外，如果不直接绑定 adapter，还可以直接监听整个列表变化，用户进入、退出和loadMoreUser() 都会触发此回调

```
    /**
     * 在线用户列表 userIn userOut信令
loadMoreUser等操作均会触发回调
     *
```

```
    * @return
    */
    Observable<List<IUserModel>>
getObservableOfOnlineUser();
```

7.3. activeUser

activeUser 为发言用户，UI 表现为独立的拉流窗口。

```
    /**
    * 用户上台监听
    * @return
    */
    Observable<LPResRoomUserInModel>
getPublishSubjectOfActiveUserAdd();

    /**
    * 使指定用户上台
    * @param userModel
    */
    void requestAddActiveUser(IUserModel
userModel);

    /**
    * 用户下台监听
    * @return
    */
    Observable<LPResRoomUserInModel>
getPublishSubjectOfActiveUserRemove();

    /**
    * 使指定用户下台
    * @param userModel
    */
```

```
23.    void requestRemoveActiveUser(IUserModel
userModel);
24.
25.    /**
26.     *  使指定用户上台被拒绝监听
27.     * @return
28.     */
29.    Observable<LPResRoomUserInModel>
getPublishSubjectOfActiveUserAddDeny();
30.
31.    /**
32.     * 是否是台上用户
33.     * @param userModel
34.     * @return
35.     */
36.    boolean isActiveUser(IUserModel userModel);
37.
38.    /**
39.     * 台上人员
40.     *
41.     * @return
42.     */
43.    List<IUserModel> getActiveUserList();
44.
45.    /**
46.     * 台下人员
47.     *
48.     * @return
49.     */
50.    List<IUserModel> getUnActiveUserList();
```

7.4. 踢人与黑名单

```
1. /**
```

```
    * 被请出教室监听
    * errorCode=1 主动踢出, errorCode=2 试听结束/教室强制关闭, errorCode=3 主讲人离开
    errorCode=6 教室结束超过30min
    */
   Observable<LPError> getObservableOfKickOut();

   /**
    * 踢出指定人员
    *
    * @param userId
    */
   void requestKickOutUser(String userId);

   /**
    * @param userId
    * @param isBlock 是否拉黑
    */
   void requestKickOutUser(String userId, boolean isBlock);

   /**
    * 移出某个黑名单用户
    *
    * @param userNumber
    */
   void freeBlockedUser(String userNumber);

   /**
    * 将所有用户解禁黑名单
    */
   void freeAllBlockedUser();

   /**
    * 黑名单成员
```

```
34.     *
35.     * @return
36.     */
37.    List<IUserModel> getBlockedUserList();
38.
39.    /**
40.     * 拉黑回调
41.     *
42.     * @return
43.     */
44.    Observable<LPResRoomBlockedUserModel>
getObservableOfBlockedUser();
45.
46.    /**
47.     * 获取当前所有的黑名单用户列表
48.     *
49.     * @return 当前所有的黑名单用户
50.     */
51.    Observable<List<IUserModel>>
getObservableOfBlockedUserList();
```

7.6. 用户分组

```
1. /**
2.     * 获取分组信息
3.     */
4.    void requestGroupInfoReq();
5.
6.    /**
7.     * 组信息更新
8.     */
9.    Observable<List<LPGroupItem>>
getObservableOfOnGroupItem();
10.
```

```
    /**
     * 获取分组集合
     *
     * @return
     */
    List<LPGroupItem> getGroupList();

    SparseArray<LPGroupItem> getGroupMap();

    /**
     * 是否是音视频分组
     *
     * @return
     */
    boolean isMediaGroup();

    /**
     * 是否显示分组
     *
     * @return true  分组显示
     * false 不分组显示
     */
    boolean enableGroupUserPublic();

    /**
     * 是否只显示本组用户列表
     *
     * @return true 显示本组 false显示全部分组
     */
    boolean enableMyGroupUsersPublish();
```

7.7. 其它

```
/**
```

```
     * 通过userId获取UserModel
     *
    * @param userId
     * @return
     */
    IUserModel getUserById(String userId);

    /**
     *  通过userNumber获取UserModel
     * @param userNumber
     * @return
     */
    IUserModel getUserByNumber(String
userNumber);

    /**
     * 获取在线用户成员
     *
     * @return
     */
    List<IUserModel> getOnlineUserList();

    /**
     * 获取学生列表
     *
     * @return
     */
    List<LPUserModel> getStudentList();

    /**
     * 获取私聊用户
     *
     * @return
     */
    List<IUserModel> getPrivateUser();
```

8. 课件与画笔 (PPTView)

Core SDK 提供了一个 `PPTView` 来展示 PPT 课件及画笔，支持多种 PPT 交互及画笔交互，自适应静态 PPT 和动态 PPT（动态 PPT 指包含 PowerPoint 动画效果的课件，使用 WebView实现）切换。将复杂的手势交互逻辑及多种可定制图形绘制封装起来方便集成开发使用。

8.1. 使用 PPTView

```
// 或在布局文件中创建
PPTView pptView = new PPTView(context);
// 3.18.2 优化静态课件翻页效果需要感知生命周期的能力，必须调用。
void addLifecycle(lifecycle);
// 注入 liveRoom
void attachLiveRoom(liveRoom);
// 销毁时需要手动调用:
void destroy();
```

8.2. 设置 PPTView 属性

```
    /**
     * 注意：可能在任意时刻调用这个方法
     *
     * @param animPPTEnable true 切为动态PPT，false切为静态PPT
     * @return true 切换成功， false 切换失败
     */boolean setAnimPPTEnable(boolean animPPTEnable)

    /**
     * 当前 PPTView 的类型，静态 or 动态
```

```
     *
     * @return
     */
    LPConstants.PPTViewType
getTargetPPTViewType()

    /**
     * 设置 PPTView 是否可以手势滑动
     * @param isFlipable
     */
    void setFlingEnable(boolean isFlipable)

    /**
     * 隐藏课件页码，默认显示。eg: 1/20
     */
    void hidePageView()
    /**
     * 显示课件页码
     */
    void showPageView()

    /**
     * 设置课件翻页权限
     *
     * @param pptAuth
     */
    void setPPTTurnPagesAuth(boolean pptAuth)

    /**
     * 获取当前PPT页数
     */
    int getCurrentPageIndex()

    /**
     * 获取PPT总页数
     */
```

```
45.    int getTotalPageNumber()
```

8.3. PPT 编辑模式

PPTView 提供3种编辑模式（EditMode，即PPTView处理触摸事件的模式）：

- NormalMode PPT滑动翻页，PPT缩放；
- ShapeMode 画笔支持绘制多钟图形。
- SelectMode 点选、框选、移动或缩放画笔；

可以通过如下方法设置，

```
1.  /**
2.  * 设置PPT编辑模式
3.  * @param pptEditMode
4.  */
5.  void setPPTEditMode(LPConstants.PPTEditMode
    pptEditMode);
6.
7.  /**
8.  * 获取PPT编辑模式
9.  * @return
10. */
11. LPConstants.PPTEditMode getPPTEditMode();
```

8.4. 翻页

NormalMode 模式下，PPT 将触摸事件处理为翻页和所缩放。静态PPT使用 viewpager 实现滑动翻页，动态 PPT 可以翻页但暂无滑动效果。

```
1.    /**
```

```
    * 翻到下一页
    */
   void gotoNextPage()
   /**
    * 翻回上一页
    */
   void gotoPrevPage()
   /**
    * 切换指定文档的指定页面
    *
    * @param docId   文档docId
    * @param pageNum 页码
    */
   void switchPPTPage(String docId, int pageNum)
```

注：学生端主动滑动PPT翻页的逻辑是不大于老师PPT的当前页面；一旦老师翻页了，学生会立即同步到老师的页面。

8.5. 画笔模式（ShapeMode ）

画笔支持绘制多钟图形。

在此模式下，PPT将触摸事件处理成画笔的绘制。画笔绘制提供多达12种画笔类型，其中包括任意曲线(Doodle)、文字(Text)、激光笔(Point)、直线(StraightLine)、单箭头(Arrow)、双箭头(DoubleArrow)、空心矩形(Rect)、实心矩形(RectSolid)、空心椭圆(Oval)、实心椭圆(OvalSolid)、空心等边三角(Triangle)和实心等边三角(TriangleSolid)。

API接口如下：

```
/**
* 绘制定制图形
* @param shapeType
*/
```

```
void setCustomShapeType(LPConstants.ShapeType shapeType);

/**
* 设置Doodle画笔线宽  仅绘制Doodle（任意曲线）时生效
* @param strokeWidth
*/
void setShapeStrokeWidth(float strokeWidth);

/**
* 设置定制图形 线宽    绘制除Doodle外其他图形时生效
* @param strokeWidth
*/
void setCustomShapeStrokeWidth(float strokeWidth);

/**
* 设置画笔颜色
* @param paintColor
*/
void setPaintColor(int paintColor);

/**
* 设置编辑文字大小 (12 px -- 80 px) 仅绘制文字时生效
* @param textSize
*/
void setPaintTextSize(int textSize);

/**
 *  清除当前页面所有画笔
 */
void eraseAllShapes();
```

8.6. SelectMode 点选、框选、移动或缩放画笔

在此模式下，PPT将触摸事件处理成画笔的点选和框选。对于选中的画笔可以进行移动、缩放和删除。

```
    /**
     * 删除选中的shape
     */
    void eraseShapes();

    /**
     * 擦除当前页所有的画笔
     */
    void eraseAllShapes();
```

8.7 多白板

PPTView 默认只有一页白板，允许动态添加多页白板和删除白板。

```
  /**
     * 添加一页白板
     *
     * @return true    添加完成
     * false   添加失败
     */
    void addPPTWhiteboardPage();

    /**
     * 删除指定pageId白板
     *
     * @param pageId 删除白板的pageId
```

```
     * @return true    删除完成
     * false   删除失败
     */
    void deletePPTWhiteboardPage(int pageId);

    /**
     * 是否支持多白板功能
     *
     * @return true    支持
     * false   不支持
     */
    boolean isMultiWhiteboardEnable();
```

8.8 设置监听

```
    /**
     * 设置PPT状态监听
     * @param pptStatusListener
     */
    public void
setPPTStatusListener(PPTStatusListener
pptStatusListener)
```

PPTStatusListener 接口定义如下，其中重点关注 `onPPTError` 回调。

`LPError.CODE_ERROR_ANIMPPT_LOAD_TIMEOUT` 代表动效课件加载超时，这时可以调用 `setAnimPPTEnable(false)` 强制切换为静态课件，保证用户能正常上课。

```
public interface PPTStatusListener {

    /**
```

```
    * ppt初始化回调，这里设置属性才生效
    */
   void onPPTViewAttached();

   /**
    * 翻页回调
    */
   void onPageChange(LPAnimPPTPageChangeEndModel pageChangeEndModel, String remarkInfo);

   /**
    * h5课件页码信息回调
    */
   void onH5PageCountChange(String docId, int page);

   /**
    * PPT 错误回调
    * @param lpError
    */
   void onPPTError(LPError lpError);

   /**
    * 动态课件开始加载回调，对应 webview 的 onPageStarted
    */
   void onAnimPPTLoadStart();

   /**
    * 动态课件加载成功回调，对应 webview 的 onPageFinish
    */
   void onAnimPPTLoadFinish();

}
```

9. 文件管理（DocListVM）

大班课班型下不用关心 9.1 ~ 9.3 节，SDK 内部默认处理了。

9.1. 拉取历史文档信息

进房间后可以通过 `liveRoom.getDocListVM().requestDocAllReq()` 拉取历史的全部课件信息。

注意 `PPTView.attachLiveRoom(liveRoom)` 时内部已经主动调用了 `requestDocAllReq()`，原则上外部不需要再次调用。

全部文档信息回调如下：

```
1.    /**
2.     获取全部文档信息回调，为 requestDocAllReq() 的返回值
3.     * @return
4.     */
5.    Observable<LPResRoomDocAllModel> getObservableOfDocAll();
```

返回的 model 字段定义如下：

```
1.  public class LPResRoomDocAllModel extends LPResRoomModel {
2.    /**
3.     * 当前显示的文档 ID
4.     */
5.    @SerializedName("doc_id")
6.    public String docId;
7.    /**
```

```
     * 当前文档的页码
     */
    @SerializedName("page")
    public int page;
    /**
     * 当前页的 step，对应动态课件的动画步数
     */
    @SerializedName("step")
    public int step;
    /**
     * 全部的文档，每一个 LPDocumentModel 代表一个文档
     */
    @SerializedName("doc_list")
    public List<LPDocumentModel> docList;
    /**
     * 当前显示的白板 id
     */
    @SerializedName("page_id")
    public int pageId;

    @SerializedName("original_class_id")
    public String originalClassId;
}
```

大班课默认会把全部的文档合并显示，即 `PPTView` 内部可以无缝跨多个文档切换翻页。专业小班课则每个文档单独一个 `PPTView`，多个 `PPTView` 之间通过 `pptView.setDocId(docId)` 来区分，大班课合并类型的 docId 强制为 “0”。

9.2. 文档上传与转码

上传

ProgressModel 为上传百分比，LPUploadDocModel 为文件上传成功后的 model。这里需要重点关注的是 `uploadDocModel.fid`，后续操作都需要用到。

```
    /**
     * 上传图片文档并转码
     * @param imagePath 图片路径
     * @return
     */
Observable<LPResponseWithProgressMergedModel
 LPUploadDocModel>>
uploadImageWithProgress(String imagePath);

   /**
     * 上传文档并转码
     * @param pptPath   文档路径
     * @param isAnimPPT 是否上传为动效课件
     */
Observable<LPResponseWithProgressMergedModel
 LPUploadDocModel>>
uploadPPTWithProgress(String pptPath, boolean
isAnimPPT);
```

查询转码进度

```
 /**
     * 获取转码进度
     * @param fid 课件 id
     * @return
     */
   Observable<LPDocTranslateProgressModel>
requestTransferProgress(String fid)
```

9.3. 文档管理

添加到教室

经 9.2 步骤上传转码成功后即可添加到教室，教室内的所有人会接收到 `getObservableOfDocAdd()` 回调刷新课件。

```
/**
    * 添加图片作为一个单独的文档
    * @param documentModel 图片 model
    * @return null 表示成功，非空为权限问题
    */
   LPError addPictureDocument(LPDocumentModel documentModel);

    /**
    * 添加文档，发 doc_add_trigger，教室内用户会收到 getObservableOfDocAdd() 回调
    * @param uploadDocModel 转码后的文档对象
    * @return null 表示成功，非空为权限问题
    */
   LPError addDocument(LPUploadDocModel uploadDocModel);
```

一个完整的上传转码、查询转码进度、添加到教室的伪代码流程如下：

```
liveRoom.getDocListVM().uploadPPTWithProgress(St
 filePath, bool isAnimPPT)
                    .subscribe(new Consumer<LPUploadDocModel>(mod) {
                        @Override
                        void onResponse(LPUploadDocModel
```

```
lpUploadDocModel) {
                        String fid =
String.valueOf(lpUploadDocModel.fileId);
                        // 每2s轮询
                        Disposable
disposableOfInterval = Observable.interval(2,
TimeUnit.SECONDS)
                                .subscribe(aLong -> {
                                    // 检测到转码成功，添加
到教室
liveRoom.getDocListVM().requestTransferProgress(n
() ->
liveRoom.getDocListVM().addDocument(lpUploadDo
                                });
                    }
                });
```

删除课件

```
/**
     * 删除Document，发 doc_del_trigger，教室内用
户会收到 getObservableOfDocDelete() 回调
     * @param docId 待删除的文档 ID
     * @return null 表示成功，非空为权限问题
     */
    LPError deleteDocument(String docId);
```

课件增加、删除回调

```
    /**
     * 新增课件回调
     * @return
     */
```

```
    Observable<LPDocumentModel>
getObservableOfDocAdd();

    /**
     * 删除课件回调
     * @return
     */
    Observable<String>
getObservableOfDocDelete();
```

专业小班课场景下，`getObservableOfDocDelete` 回调里需要把对应 docId 的 PPTView 窗口移除。

9.4. 音视频文档

音视频文档可以当做普通的文档类型添加到文件列表中，暂时只支持展示，播放需要在 PC 端点击播放。

```
/**
     * 获取媒体课件
     * @return
     */
    List<LPMediaCoursewareModel>
getMediaCoursewareList();

    /**
     * 媒体课件变化监听
     * @return
     */

Observable<List<LPMediaCoursewareModel>>
getObservableOfMediaCoursewareList();

    /**
     * 删除音视频课件
```

```
     * @param fid          文件id
     * @return
     */
    Observable<Boolean>
requestDeleteMediaCourseware(String fid);
```

9.5. 作业

作业是一种特殊类型的文档，和文档的操作基本一致，也分为上传、转码、添加到直播间、拉取历史作业。除此之外可以动态授权学生上传作答完后的文档，老师也可以把作业添加成授课用的课件，用作直播间讲解。

9.5.1 拉取历史作业

```
  /**
     * 请求首页作业列表，从老师的第一条开始，注：该操作会清除之前的所有作业列表
     * @param keyword 如果有 keyword 字段，则为搜索结果，否则是请求所有
     */
    void requestHomeworkAllList(String keyword);

  /**
     * 作业列表变更监听
     */
    Flowable<LPResHomeworkAllModel>
getObservableOfHomeworkListChanged();

    /**
     * 作业列表搜索结果监听
     */
    Flowable<LPResHomeworkAllModel>
getObservableOfHomeworkSearchRstListChanged()
```

```
16.
17.  /**
18.     * 获取当前作业列表，requestHomeworkAllList("") 后才有值
19.     * @return
20.     */
21.    LPResHomeworkAllModel getHomeworkModelList();
```

9.5.2. 作业上传与转码

上传

```
1. /**
2.     * 上传并转码作业
3.     * @param pptPath   作业的文件路径
4.     * @param userModel 上传者的用户信息
5.     * @return
6.     */
7.  Observable<LPResponseWithProgressMergedModel LPUploadHomeworkModel>> getObservableOfUploadHomeworkWithProgress(Str pptPath, LPUploadHomeworkUserModel userModel);
```

查询转码进度

同 9.2

9.5.3 作业管理

作业转码成功后可以添加为作业类型，也可以直接添加为直播课件。

添加为作业

```
/**
    * 添加可预览的作业
    */
   LPError addHomework(LPHomeworkModel
homeworkAddModel);
```

添加为课件文档

```
/**
    * 把作业转到课件区
    * @param homeworkModel 作业数据
    */
   LPError
addDocument(LPUploadHomeworkModel
homeworkModel);
```

删除作业

```
   /**
    * 删除作业
    * @param homeworkId 作业 id
    * @param userModel 用户信息
    */
   void deleteHomework(String homeworkId,
LPUploadHomeworkUserModel userModel);
```

下载作业

```
/**
    * 下载作业
    * @param homeworkId 作业文档 id
    * @param file 下载目标文件的文件夹
```

```
     * @return
     */
    Observable<ProgressModel>
downloadHomework(String homeworkId, File file);
```

10. 聊天（ChatVM）

可通过 `liveRoom.getChatVM()` 获得 `ChatVM` 引用。

10.1 公聊

10.1.1 文字消息

```
/**
     * 发送普通公聊文字消息
     *
     * @param message 消息正文
     */
    void sendMessage(String message);

    /**
     * 发送公聊文字消息，带@
     *
     * @param message 消息正文
     * @param atUserList 被艾特的用户集合
     */
    void sendMessage(String message,
Set<LPMessageAtUserModel> atUserList);

    /**
     * 发送文字消息，带引用
     *
     * @param message 消息正文
     * @param referenceModel 被引用的消息 model
```

```
    */
    void sendMessage(String message, LPMessageReferenceModel referenceModel);

    /**
     * 发送文字消息，带引用，带@
     *
     * @param message 消息正文
     * @param referenceModel 被引用的消息 model
     * @param atUserList 被艾特的用户集合
     */
    void sendMessage(String message, LPMessageReferenceModel referenceModel, Set<LPMessageAtUserModel> atUserList);
```

10.1.2. 表情消息

直播间支持的表情包由服务端返回，由 `chatVM.getExpressions()` 获取。 `LPExpressionModel` 的 key 字段为唯一标记，发送表情消息的实现为 `"[emojiKey]"` 特殊格式的文本消息。

```
/**
     * @return 服务端配置的表情列表
     */
    List<LPExpressionModel> getExpressions();

    /**
     * 发送表情消息
     *
     * @param emoji 格式为 "[emojiKey]"，emojiKey 通过 getExpressions() 中 LPExpressionModel.key
     */
    void sendEmojiMessage(String emoji);
```

```
12.
13.     /**
14.      * 发送表情消息，带@
15.      *
16.      * @param emoji
17.      * @param atUserList
18.      */
19.     void sendEmojiMessage(String emoji,
    Set<LPMessageAtUserModel> atUserList);
20.
21.     /**
22.      * 发送表情消息，带引用
23.      *
24.      * @param emoji
25.      */
26.     void sendEmojiMessage(String emoji,
    LPMessageReferenceModel referenceModel);
27.
28.     /**
29.      * 发送表情消息，带引用，带@
30.      *
31.      * @param emoji
32.      * @param referenceModel
33.      * @param atUserList
34.      */
35.     void sendEmojiMessage(String emoji,
    LPMessageReferenceModel referenceModel,
    Set<LPMessageAtUserModel> atUserList);
```

10.1.3. 图片消息

发送表情消息的实现为 `"[img:" + url + "]"` 特殊格式的文本消息，其中 url 为图片的地址。

发送图片消息分为两步，第一步上传本地图片到服务器，拿到 url 地址和图片的宽高，再调用

`liveRoom.sendImageMessage(String message, int width, int height)` 发送。

```
/**
 * 上传聊天图片
 * @param imagePath 图片路径
 * @return
 */
Observable<LPUploadDocModel> getObservableOfUploadImage(String imagePath);

/**
 * 发送图片消息
 *
 * @param message 格式为 [img:" + url + "]
 * @param width 图片宽度
 * @param height 图片高度
 */
void sendImageMessage(String message, int width, int height);
```

10.2. 私聊

```
/**
 * 是否可以私聊
 * @return
 */
boolean canWhisper();

/**
 * 获取学生私聊权限
 * 0，全部 1，老师 2助教
 *
 * @return
```

```
     */
    int getWhatRoleStudentCanPrivateChatWith();

    /**
     * 拉取私聊消息，分页加载
     *
     * @param to 私聊对象的 user_number
     * @param page 页码，默认一次加载20条
     */
    void requestWhisperList(String to, int page);

    /**
     * 私聊历史消息监听
     *
     * @return
     */
    Observable<LPWhisperListModel>
getObservableOfWhisperList();

    /**
     * 发送文字消息 —— 私聊
     * @param toUser 私聊用户 model
     * @param message 消息正文
     */
    void sendMessageToUser(@Nullable
IUserModel toUser, String message);

    /**
     * 发送表情消息 —— 私聊
     * @param toUser 私聊用户 model
     * @param emoji 格式为 "[emojiKey]"，
emojiKey 通过 getExpressions() 中
LPExpressionModel.key
     */
    void sendEmojiMessageToUser(@Nullable
IUserModel toUser, String emoji);
```

```
    /**
     * 发送图片消息 —— 私聊
     * @param toUser 私聊用户 model
     * @param message 格式为 [img:" + url + "]
     * @param width 图片宽度
     * @param height 图片高度
     */
    void sendImageMessageToUser(@Nullable IUserModel toUser, String message, int width, int height);
```

10.3. 聊天禁言

```
/**
     * 单个禁言 (仅老师助教有权限)
     *
     * @param forbidUser 被禁言的用户
     * @param duration 禁言时长，单位秒
     */
    LPError forbidChat(IUserModel forbidUser, long duration);

    /**
     * 禁言回调 (包含其他学生被禁言)
     *
     * @return
     */
    Observable<IForbidChatModel> getObservableOfForbidChat();

    /**
     * 自己被单独禁言返回 false
     * 禁言冷却时间结束了返回 true
```

```
     * @return
     */
    Observable<Boolean>
getObservableOfIsSelfChatForbid();

    /**
     * 全体禁言 (仅老师助教有权限)
     *
     * @param forbidAllChat
     */
    void requestForbidAllChat(boolean
forbidAllChat);

    /**
     * 全体禁言回调
     */
    Observable<LPRoomForbidChatResult>
getObservableOfForbidAllChatStatus();

    /**
     * 请求禁言学员信息
     */
    void requestForbidList();

    /**
     * 禁言学员列表回调
     *
     * @return
     */
    Observable<LPResRoomForbidListModel>
getObservableOfForbidList();

    /**
     * 获取禁言状态
     * @return true 表示禁言，false 未禁言
     */
```

```
51.    boolean isChatForbidden();
```

10.4. 聊天撤回

```
1.  /**
2.      * 聊天撤回
3.      * @param msgId IMessageModel.id 消息 id
4.      * @param fromId IMessageModel.from.userId 发送者 id
5.      */
6.     void requestMsgRevoke(String msgId, String fromId);
7.
8.     /**
9.      * 聊天撤回响应
10.     *
11.     * @return
12.     */
13.    Observable<LPMessageRevoke> getObservableOfMsgRevoke();
```

在接收到 `getObservableOfMsgRevoke` 回调后需要主动从消息列表的集合中移除 `LPMessageRevoke.messageId` 对应的 message，并刷新列表。

10.5. 聊天置顶

聊天置顶功能支持将某条聊天正文置顶到消息列表固定显示。

```
1.  /**
2.      * 聊天置顶
3.      *
4.      * @param message 要置顶的消息，为null时取消置顶
```

```
     */
    void requestMsgStickyList(List<IMessageModel> message);

    /**
     * 聊天置顶响应
     */
    Observable<List<IMessageModel>> getObservableOfMsgStickyList();
```

10.6. 聊天翻译

```
/**
     * 发送翻译请求
     *
     * @param message      要翻译的消息
     * @param messageId    消息id
     * @param classId      房间id
     * @param userId       自己的id
     * @param fromLanguage 待翻译的语音类型，如"zh"、"en"，也可以传 "auto"，自动识别
     * @param toLanguage   目标语言类型，如"zh"、"en"，也可以默认翻译为当前语言，Locale.getDefault().language
     */
    void sendTranslateMessage(String message, String messageId, String classId, String userId, String fromLanguage, String toLanguage);

    /**
     * 翻译回调
     * @return
     */
```

```
17.    Observable<LPMessageTranslateModel>
getObservableOfReceiveTranslateMessage();
```

10.7. 接收消息监听

收到新消息后可以自行设置到数据源显示

```
1.    /**
2.     * 收到消息回调
3.     *
4.     * @return
5.     */
6.    Flowable<IMessageModel>
getObservableOfReceiveMessage();
```

或者也可以使用

```
1. int getMessageCount();
2. IMessageModel getMessage(int position);
```

来绑定您列表的adapter，并在 `liveRoom.getChatVM().getObservableOfNotifyDataChange().subscribe(consumer);` 更新列表即可

11. 录制与转码

11.1 录制

云端录制功能只有**老师**和被授权的助教角色可以调用。

```
1. /**
2.     * 当前班型是否可以开启云录制
3.     *
```

```
     * @return
     */
    boolean canCloudRecord();

    /**
     * 设置云端录制状态，结束、开始录制、暂停。仅老师和被授权的助教有权限改变录制状态
     * @param cloudRecordStatus 录制状态的枚举
     * @return LPError null 成功，非 null 则代表失败
     */
    LPError requestCloudRecord(LPConstants.CloudRecordStatu cloudRecordStatus);

    /**
     * 云端录制状态回调
     */
    Observable<LPResCloudRecordModel> getObservableOfCloudRecordStatus();

    /**
     * 主动获取云端录制状态
     */
    LPResCloudRecordModel getCloudRecordStatus();
```

其中返回的 `LPResCloudRecordModel` 的字段结构如下：

```
public class LPResCloudRecordModel {
    /**
     * 录制状态，与 LPConstants.CloudRecordStatus 一致
     */
    public int status;
    /**
```

```
     * 触发录制的用户角色
     */
    public LPConstants.LPUserType operator;

    public LPResCloudRecordModel() {

    }

    public LPResCloudRecordModel(int status) {
        this.status = status;
    }
}
```

`CloudRecordStatus` 枚举结构如下：

```
    /**
     * 云端录制状态
     */
    public enum CloudRecordStatus {
        /**
         * 结束
         */
        Stopped(0),
        /**
         * 录制中
         */
        Recording(1),
        /**
         * 暂停
         */
        Paused(2);

        int status;

        CloudRecordStatus(int status) {
```

```
        this.status = status;
    }

    public int getStatus() {
        return status;
    }
}
```

11.2. 转码

老师或助教主动触发转码，场景一般为下课时主动调用。

```
/**
 * 触发转码
 *
 * @return 返回值无需关心
 */

Observable<LPResRoomCloudRecordStartProcessin
 requestCloudRecordStartProcessing();
```

常规场景下 `liveRoom.requestCloudRecordStartProcessing()` 即可，若为长期课触发转码后要再次出发云端录制，则需要在 `requestCloudRecordStartProcessing()` 回调内部调用 `liveRoom.requestCloudRecord(LPConstant.CloudRecordStatus.Recording)` 。

获取当前转码状态

```
/**
 * 获取转码进度
 *
 * @return
 */
```

```
6.  Observable<LPPlaybackProcessStatusModel>
requestPlaybackProcessStatus();
```

`LPPlaybackProcessStatusModel` 中 status 表示转码状态，如下

```
1.  public class LPPlaybackProcessStatusModel {
2.
3.      /**
4.       * status 0:未开启云端录制 1:开启云端录制（未转码） 2:已触发云端录制转码
5.       */
6.      public int status;
7.
8.      @SerializedName("is_long_term")
9.      public int isLongTerm;
10.
11.     public boolean isLongTermClass() {
12.         return isLongTerm == 1;
13.     }
14. }
```

12. LiveRoom 其他 API

getter

```
1.      /**
2.       * 获得当前用户
3.       */
4.      IUserModel getCurrentUser();
5.
6.       /**
7.       * 获得教室老师用户
8.       *
```

```
     * @return
     */
    IUserModel getTeacherUser();

    /**
     * 获得当前主讲人
     *
     * @return
     */
    IUserModel getPresenterUser();

    /**
     * 获取当前房间模版类型
     */
    LPConstants.TemplateType getTemplateType();

    /**
     * 获取 partner 配置
     *
     * @return
     */
    LPEnterRoomNative.LPPartnerConfig
getPartnerConfig();

    /**
     * 获取房间信息
     *
     * @return
     */
    LPRoomInfo getRoomInfo();

    /**
     * 是否为上课状态
     *
     * @return
     */
```

```
    boolean isClassStarted();
```

被踢下线（单点登录）

可以监听此回调，ILoginConflictModel会返回冲突的用户在什么终端登录，被踢时也会报LPError

```
liveRoom.getObservableOfLoginConflict().observeO
.subscribe(new Consumer<ILoginConflictModel>
() {
    @Override
    public void accept(ILoginConflictModel
iLoginConflictModel) {
    }
});
```

自定义事件广播接收

```
liveRoom.getObservableOfBroadcast().observeOn(A
.subscribe(new Consumer<LPKVModel>() {
    @Override
    public void accept(LPKVModel lpkvModel) {
        String key = lpkvModel.key;
        String value = lpkvModel.value;
    }
});
```

13. 枚举/接口说明

LPResolutionType

```
public enum LPResolutionType {
    /**
```

```
     * 流畅 320*180
     */
    LOW(0),
    /**
     * 高清 640*360
     */
    HIGH(1),

    /**
     * 720P 1280*720;
     */
    _720(2),

    /**
     * 1080P 1920*1080
     */
    _1080(3),

    /**
     * 480*270（后增加的，没法保持队形）
     */
    _360(4),

    /**
     * 960*540（后增加的，没法保持队形）
     */
    _540(5);

    private int type;

    LPResolutionType(int type) {
        this.type = type;
    }

    public int getTypeValue() {
        return type;
```

```
    }
}
```

VideoMirrorMode

```
public enum VideoMirrorMode {
        // 自动模式
        AUTO_MIRROR,
        // 水平左右翻转
        HORIZONTAL_MIRROR,
        // 垂直上下翻转
        VERTICAL_MIRROR,
        // 水平和垂直同时翻转
        HORIZONTAL_VERTICAL_MIRROR
    }
```

BRTCAudioFrameListener

```
interface BRTCAudioFrameListener {
    /**
     * 本地采集并经过音频模块前处理后的音频数据回调
     *
     * @param audioFrame  PCM 格式的音频数据帧
     */
    void
onCapturedRawAudioFrame(BRTCDef.BRTCAudioFra
 audioFrame);

    /**
     * 本地采集并经过音频模块前处理、音效处理和混
BGM 后的音频数据回调
     *
     * @param audioFrame  PCM 格式的音频数据帧
```

```
    */
    void onLocalProcessedAudioFrame(BRTCDef.BRTCAudioF
     audioFrame);

    /**
     * 暂未实现
     * @param audioFrame
     */
    void onCustomAudioRenderingFrame(BRTCDef.BRTCAudi
     audioFrame);
}
```

LPPlayerListener

```
public interface LPPlayerListener {

    /**
     * 首帧视频到达即将播放
     *
     * @param mediaId
     */
    void onReadyToPlay(String mediaId);

    /**
     * 播放音频成功
     *
     * @param mediaId
     */
    void onPlayAudioSuccess(String mediaId);

    /**
     * 播放视频成功
```

```
19.     *
20.     * @param mediaId
21.     */
22.    void onPlayVideoSuccess(String mediaId);
23.
24.    /**
25.     * 关闭流
26.     *
27.     * @param mediaId
28.     */
29.    void onPlayClose(String mediaId);
30. }
```

下载为pdf格式