Android 直播 Core SDK

- **gitlab链接**
- **javaDoc链接**
- platform: API 21+
- cpu: ARM, ARMv7a, ARM64v8a
- IDE: **Android Studio** Recommend
- **Change Log)**
- 1.x升级注意事项；1.x版本文档地址
- 2.0升级3.0 API 变更说明文档地址

功能介绍

百家云直播 Android SDK提供了 Core SDK (liveplayer-sdk-core)、开源大班课UI (BJLiveUI-Android)和 开源专业小班课UI (professional-group-class)。

- 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

```
maven { url 'http://git2.baijiashilian.com/open-android/maven/raw/master/' }
```

2.2. 添加依赖

最新版本请点击自取，**Change Log](Releases · Open Android / BJLiveCore-Android · GitLab)**

```
dependencies {
    implementation 'com.baijiayun.live:liveplayer-sdk-core:3.24.0'
}
```

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. 设置客户专属域名前缀

专属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。

```
1. 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

```
1. 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。

API 说明

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().subscrib
// 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. 教室管理

3.1. 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，通过参加码、签名和通过 url，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;

    /**
     * 用户角色
```

```
     */
    public LPConstants.LPUserType userType;

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

LPSignEnterRoomModel 字段说明

```
/**
     * 房间号
     */
    public long roomId;

    /**
     * 用户名称
     */
    public String userName;

    /**
     * 用户唯一编号
     */
    public String userNumber;

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

    /**
     * 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/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;
```

3.2. 进入房间回调说明

```
LPLaunchListener {
    @Override
    public void onLaunchSteps(int step, int totalStep) {
        //进直播间初始化任务队列回调。因为涉及信令与聊天服务器的连接，进直播间时间可能会比较长，可以根据step/totoalStep实现友好的loading效果
    }

    @Override
    public void onLaunchError(LPError error) {
        //进直播间错误回调
    }

    @Override
    public void onLaunchSuccess(LiveRoom liveRoom) {
        //进入直播间成功，返回LiveRoom对象
    }
};
```

3.3. 离开直播间

一般在 **Activity**的 `onDestroy()` 中调用

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

3.4. 其它 API

4. 本地推流

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

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

/**
 * 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.LPVideoVi
// 或者
cameraView.setViewType(LPConstants.LPVideoVie
// 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。

LPResolutionType 见 跳到第一个章节

4.2 推流镜像

4.3. 基础美颜

4.4. 屏幕分享

4.5. 课前预览

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

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

其中 `BRTCListener.BRTCAudioFrameListener` 接口定义如下

```
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.BRTCAudioFram
     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.BRTCAudio
     audioFrame);
}
```

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

4.7. Others

```
// 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

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，可以通过如下方法批量获取ActiveUser。

```
Observable<List<IMediaModel>> obs = liveRoom.getSpeakQueueVM().getObservableOfAct
Consumer<List<IMediaModel>> consumer = iMediaModels -> initView();

// 先订阅
obs.subscribe(consumer);
// 再发送ActiveUser请求
liveRoom.getSpeakQueueVM().requestActiveUsers
```

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) {

    }
});
```

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...
}
```

注：所有接口类型均打进了源码包sources.jar，可以在Android Studio中直接搜索类名看到该类及注释。

当获得这个用户对象的mediaId就能开始拉流，播放其音视频了。

```
LPPlayer player = liveRoom.getPlayer();
String mediaId = iMediaModel.getMediaId();
player.playAudio(mediaId);   //播放音频
player.playVideo(mediaId, videoView);   //播
放音视频
player.playAVClose(mediaId); //关闭音视频流
```

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

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

LPVideoView提供了SurfaceView和TextureView两种View进行视频渲染（如果为RTC引擎，目前仅支持SurfaceView）

```
videoView.setViewType(LPPConstants.LPVideoVie
// 或者
videoView.setViewType(LPConstants.LPVideoView
```

此外，LPPlayer还提供如下一些方法满足某些使用场景

```
1. void addPlayerListener(LPPlayerListener listener);     //增加播放音视频回调
2. void removePlayerListener(LPPlayerListener listener);  //移除播放音视频回调
3. LPConstants.LPLinkType getLinkType();  //获得下行链路类型  (RTC引擎不支持)
4. void setLinkType(LPConstants.LPLinkType linkType);     //设置下行链路类型  (RTC引擎不支持)
5. Observable<LPConstants.LPLinkType> getObservableOfLinkType(); //链路类型改变回调 (RTC引擎不支持)
6. int getCurrentUdpDownLinkIndex();  //UDP下行服务器Index  (RTC引擎不支持)
7. void setCurrentUdpDownLinkIndex(int index);  // (RTC引擎不支持)
8. LPConstants.MediaSourceType getMediaSourceType();  // 获取媒体内容类型
9. List<? extends IMediaModel> getExtraStreams();  // 获取其他流 仅在ActiveUser回调时可能不为空
```

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RTC引擎，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

```
// WebRTC中，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。

4. 举手发言

学生请求举手发言相关接口

```
liveRoom.getSpeakQueueVM().requestSpeakApply(
    // 请求举手
```

```
liveRoom.getSpeakQueueVM().requestSpeakApply(
 listener); // 支持举手倒计时回调
liveRoom.getSpeakQueueVM().cancelSpeakApply()
 // 取消举手
Observable<IMediaControlModel>
getObservableOfSpeakResponse(); // 学生监听老
师同意或者拒绝举手
```

学生请求发言后，会在百家云标准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。

学生收到老师远程控制信令

```
liveRoom.getSpeakQueueVM().getObservableOfMed
        .subscribe(new
Consumer<IMediaControlModel>() {
            @Override
            public void
accept(IMediaControlModel
iMediaControlModel) {
                if
(!iMediaControlModel.isApplyAgreed()) {
                    // 老师关闭发言
                }
            }
        });
```

如果是APP作为老师端，还可以使用以下一些方法

收到学生举手申请回调

```
Observable<IMediaModel>
getObservableOfSpeakApply();
```

同意/拒绝学生举手

```
void agreeSpeakApply(String userId);
void disagreeSpeakApply(String userId);
```

关闭他人发言，仅**老师**角色可用

```
liveRoom.getSpeakQueueVM().closeOtherSpeak(us
```

控制学生推音视频状态，仅**老师**角色可用

```
void controlRemoteSpeak(String userId,
boolean isVideoOn, boolean isAudioOn);
```

5. 在线用户

在线用户模块可以通过liveRoom.getOnlineUserVM获得，其提供了

```
int getUserCount();
IUserModel getUser(int position);
```

两个方法，可以方便高效的绑定UI的Adapter。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）

```
liveRoom.getOnlineUserVM().loadMoreUser();
// loadMoreUser 响应
liveRoom.getOnlineUserVM().getObservableOfUse
```

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

```
liveRoom.getOnlineUserVM().getObservableOfOnl
 Consumer<List<IUserModel>>() {
    @Override
    public void accept(List<IUserModel>
iUserModels) {
    }
});
```

用户进入房间（房间人数小于100人时）

```
liveRoom.getObservableOfUserIn().observeOn(An
 Consumer<IUserInModel>() {
    @Override
    public void accept(IUserInModel
iUserInModel) {
    }
});
```

用户退出房间（房间人数小于100人时）

```
liveRoom.getObservableOfUserOut().observeOn(A
 Consumer<String>() {
    @Override
    public void accept(String userId) {
    }
});
```

房间人数变化（实时）

```
liveRoom.getOnlineUserVM().getObservableOfOnL
 Consumer<Integer>() {
    @Override
    public void accept(Integer integer) {
```

```
    }
});
```

6. 课件与画笔

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达12可定制图形绘制封装起来方便集成开发使用。初始化方法如下，

```
// 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);
void attachLiveRoom(liveRoom);

// 销毁时需要手动调用:
void destroy();
```

切换PPT在容器中显示全屏\铺满

```
void setPPTShowWay(LPConstants.LPPPTShowWay.SHOW_F //全屏
void setPPTShowWay(LPConstants.LPPPTShowWay.SHOW_C //铺满
LPConstants.LPPPTShowWay getPPTShowWay() //获取显示模式
```

静态\动态PPT模式切换 （只有老师上传了动态PPT时才可以切换！）

```
 /**
 * @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(true);

 /**
 * @return 当前是否为动态PPT
 */
boolean isAnimPPTEnable()
```

PPTView常用事件监听

```
void setOnViewTapListener(new OnViewTapListener() {
            @Override
            public void onViewTap(View view, float x, float y) {
            //处理单击事件
            }
        });

void setOnDoubleTapListener(new OnDoubleTapListener() {
            @Override
            public void onDoubleTapConfirmed() {
            //处理双击事件
            }
        });


void setOnPageSelectedListener(new WhiteboardView.OnPageSelectedListener() {
```

```
            @Override
            public void onPageSelected(int position) {
                // 翻页回调;
            }
        });

void setPPTErrorListener(new OnPPTErrorListener() {
            @Override
            public void onAnimPPTLoadError(int errorCode, String description) {
                //动态PPT加载失败监听
            }
        });
```

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，

```
/**
 * 设置PPT编辑模式
 * @param pptEditMode
 */
void setPPTEditMode(LPConstants.PPTEditMode pptEditMode);

```

```
/**
 * 获取PPT编辑模式
 * @return
 */
LPConstants.PPTEditMode getPPTEditMode();
```

NormalMode 翻页及缩放

在此模式下，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。如果想要静止滑动翻页可以调用

```
void setFlingEnable(false);
boolean isFlingEnable(); //是否支持滑动
```

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。

此外PPT翻页也可以通过API调用实现（即使禁止了滑动翻页也生效）

```
/**
* 翻到PPT任意页数
*@param index 目的页数
*/
void updatePage(int index);

/**
* 翻到下一页
*/
void gotoNextPage();

/**
* 翻回上一页
*/
```

```
15. void gotoPrevPage();
```

显示/隐藏默认页码框

```
1. void showPPTPageView();
2. void hidePPTPageView();
```

其他页码相关API

```
1. /**
2. *@return 当前页数
3. */
4. int getCurrentPageIndex();
5.
6. /**
7. * 获取PPT总页数
8. * @return
9. */
10. int getTotalPageNumber();
11.
12. /**
13. *@return 当前页是否是老师/助教所在页（学生端可以调用）
14. */
15. boolean isCurrentMaxPage();
```

缩放功能静态PPT默认开启，可以通过双击或者手势放大及缩小PPT，动态PPT在暂不支持。

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)。一些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();
```

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();
```

7. 聊天

发送消息

文字消息

```
liveRoom.getChatVM().sendMessage(msg);
```

```
liveRoom.getChatVM().sendMessage(msg, channel);
```

发送表情

```
liveRoom.getChatVM().sendEmojiMessage("[" +
emoji.key + "]");
```

获取配置的表情库

```
List<IExpressionModel> expressions =
liveRoom.getChatVM().getExpressions();
```

发送图片

```
String imageContent =
LPChatMessageParser.toImageMessage(imageUrl);
liveRoom.getChatVM().sendImageMessage(imageCo
 imageWidth, imageHeight);
```

接收消息

收到新消息

```
liveRoom.getChatVM().getObservableOfReceiveMe
 Consumer<IMessageModel>() {
    @Override
    public void accept(IMessageModel
iMessageModel) {
        String channel =
iMessageModel.getChannel();
        String msg =
iMessageModel.getFrom().getName() + ":" +
iMessageModel.getContent();
    }
});
```

或者也可以使用

```
int getMessageCount();
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

8. 直播带货相关API

商品列表内变更

商品 Model 属性

```
// 商品 id
String id;

// 商品 名称
String name;

// 商品图片 url
String imgUrl;

// 商品原价
float price;

// 商品折扣后价格
float discountPrice;

// 购买链接，跳转购买的 url
String buyUrl;

// 商品简介
String desc;

// 显示顺序
```

```
23. int order;
24. 
25. // 是否已上架
26. boolean isOnShelf;
```

商品列表监听（注：学生身份仅会获取到已上架的商品）

```
1. liveRoom.getLiveShowVM().getObservableOfSellP
2.  .observeOn(AndroidSchedulers.mainThread())
3.  .subscribe(this::updateProductList)
```

商品缓存获取（注：学生身份仅会获取到已上架的商品）

```
1. liveRoom.getLiveShowVM().getSellProductsAll()
```

请求下一页商品数据，新的数据会在 商品列表监听 回调中返回

```
1. liveRoom.getLiveShowVM().requestNextPage();
```

刷新列表，数据会在 商品列表监听 回调中返回

```
1. liveRoom.getLiveShowVM().refreshProductList()
```

请求变更商品上架状态(仅老师助教可调用)，状态变更结果在商品列表监听 回调中返回

```
1. liveRoom.getLiveShowVM().requestChangeShelfSt
    shelfState);
```

获取商品总数（注：学生身份仅会获取到已上架的商品总数）

```
liveRoom.getLiveShowVM().getProductCount()
```

商品总数监听（注：学生身份仅会获取到已上架的商品总数变更）

```
liveRoom.getLiveShowVM().getObservableOfProdu
.observeOn(AndroidSchedulers.mainThread())
.subscribe(this::setTvCount)
```

点赞

进房间请求点赞数量，结果在点赞数监听中返回

```
liveRoom.getLiveShowVM().requestLiveLikeCount
```

直播点赞数变更监听

```
liveRoom.getLiveShowVM().getObservableOfLiveL
.observeOn(AndroidSchedulers.mainThread())
    .subscribe {
        // 新的点赞数
    }
```

获取当前点赞数

```
liveRoom.getLiveShowVM().getLiveLikeCount()
```

发送点赞请求

```
liveRoom.getLiveShowVM().requestSendLiveLike(
```

礼物：旧版礼物API已标记弃用，请移步下一标签查看打赏API

礼物数据 Model 说明

```
 /** 礼物 id */
int giftId;

 /** 礼物个数，若为赠送礼物发送，则表示请求赠送的个数 */
int count;

 /** 赠送人 id */
String userId;

/** 赠送人 name */
String name;

/** 赠送人 头像url */
String url;
```

赠送礼物，赠送后，本人将立即收到赠送结果，其他人将在礼物变更广播中通知

```
val giftModel = giftAdapter?.getItem(selectedPosition) //获取当前赠送礼物的id等数据
giftModel?.count = 1 // 设置赠送的礼物个数
// region 初始化赠送人的相关信息
val currentUser = routerViewModel?.liveRoom?.currentUser
giftModel?.name = currentUser?.name
giftModel?.userId = currentUser?.userId
giftModel?.url = currentUser?.avatar
```

```
// endergion
// 发送赠送礼物请求
routerViewModel?.liveRoom?.liveShowVM?.reques
```

礼物赠送结果监听

```
liveRoom.liveShowVM.observableOfGiftChange
.observeOn(AndroidSchedulers.mainThread())
    .subscribe {
        // 返回所有礼物的键值对信息
    })
```

赠送礼物事件消息通知，注：

1. 老师身份监听时，每次其他学生每次调用{@link #requestSendGift(LPLiveGiftModel)}都会立即收到此通知，返回本次操作的人员信息及礼物信息；
2. 学生身份监听时，直播送主播礼物广播消息，由服务端每10秒一次同步，返回这10秒内所有的赠送礼物人员信息及礼物信息，不包含当前登录的学生本身赠送的礼物记录。

```
liveRoom.liveShowVM.observableOfSendGift
.observeOn(AndroidSchedulers.mainThread())
    .subscribe {
        if (it.size == 1 &&
it[0].userId?.equals(liveRoom.currentUser.use
 {
            // 为自身赠送结果返回
        } else {
            // 为赠送礼物事件消息通知，需要处理
所有礼物变更消息
        }
    }
```

获取当前所有的礼物列表

```
liveRoom.liveShowVM.getGiftAll()
```

打赏

1. 获取后台打赏配置

获取打赏配置

```
liveRoom.liveShowVM.rewardConfig
```

监听后台打赏配置更新，返回打赏type对应打赏配置LPRewardDataModel的Map参数

```
liveRoom.liveShowVM.observableOfRewardConfigU
```

LPRewardDataModel说明：

```
/** 该配置是否打开 */
boolean isOpen;

/** 礼物key值：1现金打赏，2现金礼物，3积分 */
int type;

/** 礼物配置model，若为现金打赏则Object解析成String，若为礼物或积分打赏则解析成LPLiveRewardConfigModel */
List<Object> configs;

/** 现金打赏最小金额，只有配置为现金打赏时该参数才有效 */
float minMoney;
```

LPLiveRewardConfigModel说明：

```
/** 特效图片url */
String imgUrl;
/** 礼物名称 */
String name;
/** 礼物代表价格，价格0则免费；积分则代表积分个数(整值) */
float price;
/** 是否显示浮窗效果 */
boolean isFloat;
/** 是否在直播间显示 */
boolean isShow;
/** 是否显示特效 */
boolean specialEffects;
/** 礼物对应的key值 */
int rewardKey = -1;
/** 打赏人名称 */
String sendUserName;
```

1. 绑定手机号，发送免费礼物可跳过此步骤

传入当前手机号参数，请求获取短信验证码

```
liveRoom.liveShowVM.getVerificationCode(Strin
    phoneNumber)
```

校验并绑定手机号，code为上一步短信发送的验证码，绑定成功会返回**user_token**

```
liveRoom.liveShowVM.checkPhoneCode(phoneNumbe
    code)
```

请求账户余额

```
liveRoom.liveShowVM.getAccountBalance(user_to
```

1. 发起打赏。若发送免费礼物，则直接使用如下api；若发送付费礼物，需要先检查余额，若余额不足则需要进入充值流程

请求发起打赏

```
liveRoom.liveShowVM.startReward(LPRewardModel
        .filter {
            // 若是该礼物配置更新，则视为打赏失败，则通知更新配置
            return@filter if (it.isConfigChange) {
                // 后台配置已更改，触发请求后台配置

                liveRoom.liveShowVM.triggerUpdateRewardConfig
                false
            } else {
                true
            }
        }
        .subscribe({
           // 打赏成功
        }, {
           // 打赏失败
        })
```

LPRewardModel说明：

```
/** 打赏名称："现金"、礼物名称*/
String rewardName;

/** 金额/分 */
int money;
```

```
6.
7.  /** 类型，1现金红包，2礼物，3积分 */
8.  int type;
9.
10. /** 打赏图片效果，和配置返回值一致 */
11. String customImg;
12.
13. /** 打赏特效，默认0，1为有特效，和配置返回值一致
    */
14. boolean specialEffects;
15.
16. /** 浮窗效果，默认0，1为有浮窗效果，和配置返回值
    一致 */
17. boolean isFloat;
18.
19. /** money大于0时必传 */
20. String token;
```

打赏通知显示特效回调

```
1.  liveRoom.liveShowVM.observableOfSpecialEffect
```

打赏通知发送聊天消息回调

```
1.  liveRoom.liveShowVM.observableOfSendRewardMes
2.
    .observeOn(AndroidSchedulers.mainThread())
3.          .subscribe {
4.                  // 通过返回LPRewardModel构造
    LPMessageDataModel
5.                  // ...
6.                  // 调用sendMessage发送打赏消息，
    其中LPMessageDataModel中的isFloat字段表示是否
    显示浮窗效果
```

```
routerViewModel.liveRoom.chatVM.sendMessage(m
 lpMessageDataModel)
      }
```

1. 充值

请求预支付相关参数，其中appId为集成端app的微信相关appId，money的单位是分，token为手机号绑定时的user_token，返回的参数为LPRechargeParamsModel

```
liveRoom.liveShowVM.getObservableOfStartRecha
 money, token)
            .subscribe({
                // 获取成功，进入调起微信支付流程
            }, {
                // 获取失败
            })
```

LPRechargeParamsModel说明：

```
/** app唯一标识，由本地提供*/
String appId;
/** 商户号，目前只能由后端返回 */
String partnerId;
/** 预支付ID */
String prepayId;
/** 签名 */
String sign;
/** 订单编号，用于查询订单状态 */
String code;
// ...
```

调起微信支付：直播带货模版中定义有标准的微信支付调起流程实现接口WePayAPI，可以实现该接口或继承微信支付流程基类BaseWePayImpl实现自己的微信支付流程类

```
public interface WePayAPI {

    /**
     * 获取app唯一的ID，由接入端自己提供
     */
    String getAppId();

    /**
     * 注册app
     *
     * @param context activity
     * @param appId app的唯一标识
     */
    void registerApp(Context context, String appId);

    /**
     * 注册支付回调
     * @param callback
     */
    void registerPayCallback(ResultCallback callback);

    /**
     * 通过IWXApi拉起微信进行支付
     *
     * @param signModel 带签名的参数
     * @return false：调起微信失败
     */
    boolean requestWePay(SignModel signModel);
```

```
29.
30.     /**
31.      * 支付回调返回当前支付错误码
32.      */
33.     void notifyPaymentCallback(int errCode,
    String errString);
34.
35.     /**
36.      * 是否安装微信app
37.      */
38.     boolean isWxAppInstalled();
39.
40.     /**
41.      * 是否支持微信支付api
42.      */
43.     boolean isWxApiSupported();
44.
45.     /**
46.      * unregister from weixin.
47.      * If you want to use some methods, you
    may need to be in the method cycle between
    {@link #registerApp}
48.      * and {@link #unRegisterApp}
49.      */
50.     void unRegisterApp();
51. }
```

查询订单状态，其中code为预支付接口返回的code参数，token为user_token

```
1. liveRoom.liveShowVM.checkLastOrderStatus(code
    token)
2.     .subscribe ({
3.         showToastMessage(if (it) "支付成功"
    else "支付失败")
```

```
    })
```

1. 获取礼物数量，此处的礼物数量为打赏礼物的数量，不包含旧版礼物API的数量

获取打赏礼物数量

```
liveRoom.liveShowVM.giftAll2
```

打赏礼物数量更新监听

```
liveRoom.liveShowVM.observableOfGiftCountChan
```

9. 录课

云端录制功能只有**老师**角色可以调用

```
liveRoom.requestCloudRecord(ture);
// 开始录制
liveRoom.requestCloudRecord(false);
// 停止录制
Observable<Boolean>
getObservableOfCloudRecordStatus();     //
云端录制状态KVO
```

10. 公告

房间公告支持跳转，如果不需要可以直接传null。
主动获取直播间公告

```
liveRoom.requestAnnouncement();
```

设置直播间公告，仅**老师**角色可用

```
liveRoom.changeRoomAnnouncement(content,
link);
```

直播间公告变更通知

```
liveRoom.getObservableOfAnnouncementChange().
    .subscribe(new
Consumer<IAnnouncementModel>() {
        @Override
        public void
accept(IAnnouncementModel
iAnnouncementModel) {
            String content =
iAnnouncementModel.getContent();
            String url =
iAnnouncementModel.getLink();
    }
});
```

11. 测验V1

下为测验V1的API，如果需要使用旧版测验可以联系技术支持在服务器进行配置。

获取历史测验

```
liveRoom.getSurveyVM().requestPreviousSurvey(
        .subscribe(new
Consumer<IPreviousSurveyModel>() {
    @Override
    public void accept(IPreviousSurveyModel
iPreviousSurveyModel) {

iPreviousSurveyModel.getPreviousSurvey() //
```

```
历史测验List
6.
iPreviousSurveyModel.getRightCount()     //
当前用户答对几题
7.
iPreviousSurveyModel.getWrongCount()     //
当前用户打错几题
8.     }
9. });
```

收到老师发送新的测验

```
1. liveRoom.getSurveyVM().getObservableOfSurveyR
 Consumer<ISurveyReceiveModel>() {
2.     @Override
3.     public void accept(final
ISurveyReceiveModel iSurveyReceiveModel) {
4.         iSurveyReceiveModel.getSurvey()  //
新的测验
5.     }
6. });
```

学生发送答案

```
1. /**
2.  * 学生发送答案
3.  *
4.  * @param order       题目序号
5.  * @param userName    学生姓名
6.  * @param userNumber
7.  * @param answer      [A, B ,C] 数组元素是
option 下 key
8.  * @param result      0 正确 1 错误 -1 没有答
案（老师没有设置正确答案，所有的option的isAnswer
```

```
都是false）
 */
liveRoom.getSurveyVM().sendAnswer(int order, String userName, String userNumber, List<String> answer, int result);
```

服务器答题统计

服务器会10秒汇总一次，如果有答题状态更新的话就广播下发

```
/**
 * 收到测验统计结果回调
 *
 * @return
 */
Observable<ISurveyStatisticModel> getObservableOfAnswerStatistic();
```

模型接口说明

```
ISurveyModel {
    int getOrder();
//题目序号
    String getQuestion();
//获取题干
    List<ISurveyOptionModel> getOptionList();   //各个选项
}
ISurveyOptionModel{
    String getKey();
//获得选项标识 A,B,C \ 1,2,3 ...
    String getValue();
//获得选项值
    boolean isAnswer();
//是否是正确答案
```

```
    int getUserCount();
//该选项选择人数
}
ISurveyStatisticModel{
    int getOrder();
//题目序号
    Map<String, Integer> getResult();
//获得统计结果  key 是 option key，  value 是选择的人数
}
```

可以参考UI实现

12. 测验V2

新版测验接口在 `QuizVM` 类中，调用 `liveRoom.getQuizVM`

测验v2信令

发布答题

```
/**
 * 发布答题
 *
 * @param quizId
 * @param forceJoin true：强制答题 false：不强制答题
 */
void requestQuizStart(String quizId, boolean forceJoin);
```

转发

```
/**
```

```
 * 服务端转发开始答题
 * @return
 * LPJsonModel：
 * {
 *     message_type: "quiz_start",
 *     quiz_id: {string},
 *     force_join: {number} // 0 不强制答题 1 强制答题
 * }
 */
Observable<LPJsonModel>
getObservableOfQuizStart();
```

终止答题

```
/*
 * 终止答题
 * @param quizId
 */
void requestQuizEnd(String quizId);
```

转发终止答题

```
/**
 * 服务端转发终止答题
 * @return
 * LPJsonModel:
 * {
 *     message_type: "quiz_end",
 *     quiz_id: {string}
 * }
 */
Observable<LPJsonModel>
getObservableOfQuizEnd();
```

老师公布答案

```
/**
 * 老师公布答案
 *
 * @param quizId
 * @param solution 后面参数的solution的map都是如下形式：
 * {
 *     "123": 1,// question_id => solution
 *     "124": [12, 13],
 *     "125": "长江",
 * }
 */
void requestQuizSolution(String quizId, Map<String, Object> solution);
```

服务器转发答案

```
/**
 * 服务端转发答案
 * @return
 * LPJsonModel：
 * {
 *     message_type: "quiz_solution",
 *     quiz_id: {string},
 *     solution: {
 *         "123": 1,// question_id => solution
 *         "124": [12, 13],
 *         "125": "长江",
 *     }
 * }
 */
```

```
Observable<LPJsonModel>
getObservableOfQuizSolution();
```

请求当前正在答的题

```
/**
 * 当前正在答的题
 *
 * @return
 */
Observable<LPJsonModel>
getObservableOfQuizRes();
```

当前正在答的题

```
/**
 * 当前正在答的题
 * @return
 * LPJsonModel：
 * {
 *     message_type: "quiz_res",
 *     quiz_id: {string},     // 如果当前无答题，则quiz_id为空字符串
 *     solution: {
 *         "123": 1,// question_id => solution
 *         "124": [12, 13],
 *         "125": "长江",
 *     }, // 如果曾经提交过，返回之前提交的结果，否则为空
 *     force_join: {number}, // 0 不强制答题 1 强制答题
 *     end_flag: {number}  // 0 未触发结束答题，1 已触发结束答题
 * }
```

```
 */
Observable<LPJsonModel>
getObservableOfQuizRes();
```

学生答题

```
/**
 * 学生答题
 *
 * @param quizId
 * @param solution
 */
void submitQuiz(String quizId, Map<String,
Object> solution);
```

学生答题转发给老师和助教

```
/**
 * 学生答题发给老师或助教
 *
 * @param quizId
 * @param solution
 */
void submitQuizToSuper(String quizId,
Map<String, Object> solution);
```

测验v2接口

获取试卷列表(老师)

```
/**
 * 获取试卷列表
 *
 * @return
 *LPQuizModel中仅有quizId和title
```

```
 */
Observable<List<LPQuizModel>>
getListQuiz();
```

新建/更新试卷(老师)

```
/**
 * 新建/更新试卷
 *
 * @param lpQuizModel 新建quiz_id和
question_id和option_id为0
 * @return
 */
LPError saveQuiz(LPQuizModel lpQuizModel);
```

删除试卷(老师)

```
/**
 * 删除试卷
 *
 * @param quizId
 * @return
 */
Observable<Boolean> deleteQuiz(long
quizId);
```

获取试卷详情/答题详情(老师)

```
/**
 * 获取试卷详情/答题详情
 *
 * @param quizId
 * @param type   0试卷详情1答题详情
 * @return
 */
```

```
Observable<LPQuizModel> getQuizDetail(long quizId, LPConstants.LPExamQuizType type);
```

导入试卷(老师)

```
/**
 * 导入试卷  excel文件
 *
 * @param excelPath
 * @return
 */
Observable<Boolean> importExcel(String excelPath);
```

获取试卷导出地址(老师)

```
/**
 * 获取试卷导出地址
 * @param quizId
 * @param type 0 导出试卷 1导出测验结果
 * @return
 */
Observable<String> getExportUrl(long quizId, LPConstants.LPExamQuizType type);
```

学生接口获取试卷(学生)

```
/**
 * 学生接口获取试卷
 * @param quizId
 * @return
 */
Observable<LPQuizModel> getQuizInfo(long quizId);
```

测验广播列表

```
/**
 * 测验广播列表
 *
 * @return
 */
LPQuizCacheModel getQuizCacheList();
/**
 * 获取测验广播列表
 *
 * @return
 */
Observable<LPQuizCacheModel>
getObservableOfQuizCacheList();
```

大小班小测同步信息请求

```
/**
 * 大班课小测同步信息(大班切到小班)
 */
void requestRoomQuiz();
```

大小班小测同步信息响应

```
/**
 * 大班课小测同步信息
 * @return
 * {
 *    "quiz_id":{string},
 *    "quiz_title":{string}
 * }
 */
Observable<List<LPQuizModel>>
getObservableOfRoomQuiz();
```

13. 问答

目前移动端只支持学生端问答，请求历史问答：

UI SDK入口需在百家云后台配置才显示，配置项为 **enable_live_question_answer**

```
/**
* 请求历史问答
*
* @return LPError {@link LPError#CODE_ERROR_INVALID_ARGUMENT} 没有更多问题信息
* Deprecated 请使用 {@link ToolBoxVM#requestQuestionPullReq(com.baijiayu
*/
@Deprecated
LPError loadMoreQuestions();
/**
* 是否有更多问题页数
*
* @return boolean
*/
boolean isHasMoreQuestions();
```

学生发送问题：

```
/**
* 发送问题
*
* @param content 提问问题
* @return LPError
* {@link LPError#CODE_ERROR_INVALID_ARGUMENT} 输入内
```

```
容错误;
 * {@link LPError#CODE_ERROR_QUESTION_SEND_FORBID} 权限错误
 * Deprecated 请使用 {@link ToolBoxVM#requestQuestionSend(java.lang.Strin
 */
@Deprecated
LPError sendQuestion(String content);
```

问答队列监听

```
/**
 * 问答 返回问答队列
 * @return List<LPQuestionPullResItem>
 * Deprecated   参考{@link ToolBoxVM#getObservableOfQuestionSendRes()
 *                  @link ToolBoxVM#getObservableOfQuestionPub()
 *                  @link ToolBoxVM#getObservableOfQuestionPullRes()}
 */
@Deprecated
Observable<List<LPQuestionPullResItem>> getObservableOfQuestionQueue();
```

模型接口说明

```
LPQuestionPullResItem {
    int status;
// 已发布 1,未发布 2, 已回复 4,未回复 8,全部 15

    boolean forbid;
// 是否禁止该学生提问
```

```
    String id;
// 问题id

    List<LPQuestionPullListItem> itemList;
// 问题和回复列表 （问题和老师追加的回复）
}

LPQuestionPullListItem {
    long time;
//发布时间戳

    String content;
//发布内容

    LPUserModel from;
//发布人信息
}
```

可以参考UI实现

- 以上接口废弃，推荐使用新版问答，接口在`ToolBoxVM`，调用`liveRoom.getToolBoxVM`

发送问题

```
/**
 * 发送问题
 *
 * @param content 提问问题
 * @return LPError
 * {@link LPError#CODE_ERROR_INVALID_ARGUMENT} 输入内容错误;
```

```
 * {@link LPError#CODE_ERROR_QUESTION_SEND_FORBID} 权限错误
 */
LPError requestQuestionSend(String content);

/**
 * 获取问答回调
 */
Observable<LPQuestionSendModel> getObservableOfQuestionSendRes();
```

获取历史问答

```
/**
 * 请求历史问答
 *
 * @return LPError {@link LPError#CODE_ERROR_INVALID_ARGUMENT} 没有更多问题信息
 */
LPError requestQuestionPullReq(LPQuestionPullReqModel lpQuestionPullReqModel);

/**
 * 历史问答
 */
Observable<LPQuestionPullResModel> getObservableOfQuestionPullRes();
```

发布、取消、回复问答

```
/**
 * 发布,取消发布,回复问答
 */
void
requestQuestionPub(LPQuestionPubTriggerModel
 lpQuestionPubTriggerModel);

/**
 * 发布,取消发布,回复问答  返回
 */
Observable<LPQuestionPubModel>
getObservableOfQuestionPub();
```

是否有问答权限

```
/**
 * 是否有问答权限
 */
Observable<Boolean>
getObservableOfQuestionForbidStatus();
```

可以参考UI实现

14. 答题器

答题器接口在 `ToolBoxVM` 类中，调用 `liveRoom.getToolBoxVM`

数据结构说明

LPAnswerModel

老师发布答题器及答题器响应相关的model，发布答题器时需要构造LPAnswerModel。发布答题器需要的参数如下：

```
// 0/选择题 1/判断题
public int type;
// 答题时长，秒
public long duration;
// 题目描述
private String description;
// 答题开始的unix时间戳，即服务端收到answer_start_trigger的时间
public long timeStart;
// 答题完成时是否显示正确答案
public boolean isShowAnswer;
// 选项
public List<LPAnswerSheetOptionModel> options;
```

LPAnswerSheetOptionModel

表示每个选项的信息，发布答题器和回答时都会用到。其中每个字段含义如下

```
// 选项名，必填
public String text;
// 是否是正确选项
public boolean isRight;
//旧版本信令字段，同isRight
public boolean isCorrect;
// 是否选择了该选项，对单多选、判断都适用。默认false，回答时必填
public boolean isActive;
// 该选项被选择的次数
public int selectedCount;
```

LPAnswerRankModel

包含一个rankList参数，表示当前分组的排名。item为AnswerRankContent表示当前分组参与答题的用户，如下所示：

```
// 分组id
public int groupId;
// user_id
public String userId;
// user_number
public String userNumber;
// 人名
public String userName;
// 排名
public int rank;
```

发布答题（触发）

```
/**
 * 发布答题（触发） 一般由老师调用
 *
 * @param lpAnswerModel
 */
void requestAnswerStart(LPAnswerModel lpAnswerModel);
```

发布答题（响应）

```
/**
 * 发布答题（响应）
 *
 * @return
 */
Observable<LPAnswerModel> getObservableOfAnswerStart();
```

停止/撤销答题（触发）

```
/**
 * 停止/撤销答题（触发）
 * @param isRevoke 是否是撤销
 * @param delay    延时结束
 */
void requestAnswerEnd(boolean isRevoke,
long delay);
```

停止/撤销答题（响应）

```
/**
 * 停止/撤销答题（响应）
 */
Observable<LPAnswerEndModel>
getObservableOfAnswerEnd();
```

答题数据更新

```
/**
 * 答题数据更新（如有学生答题即可收到更新）
 */
Observable<LPAnswerModel>
getObservableOfAnswerUpdate();
```

请求历史答题数据（请求）

```
/**
 * 请求历史答题数据（请求）
 * @param id 传id则返回某次的答题数据，传""则返
回本节课所有答题历史数据
 */
```

```
void requestAnswerPullReq(String id);
```

请求历史答题数据（返回）

```
/**
 * 请求历史答题数据（返回）
 * @return map的key为某次答题的id
 */
Observable<Map<Object,
LPAnswerRecordModel>>
getObservableOfAnswerPullRes();
```

提交答案

```
/**
 * 提交答案
 */
boolean submitAnswers(LPAnswerModel
lpAnswerModel);

/**
  * 提交答案
  * @params timeUsed 本地设置答题用时
  */
boolean submitAnswers(LPAnswerModel
lpAnswerModel, long timeUsed);
```

请求答题器排名数据

```
/**
 * 请求答题器排名数据
 *
 * @param top 前top名
 * @param userNumber 自己的userNumber
```

```
 * @param groupId 自己的分组id、id为0表示未分组或者是大班身份
 */
void requestAnswerRankList(int top, String userNumber, int groupId);
```

答题器排名数据返回

```
/**
 * 答题器排名数据返回
 */
Observable<LPAnswerRankModel> getObservableOfAnswerRankRes();
```

15. 点赞

助教和老师可以给学生点赞，学生无法点赞，助教和老师不能被点赞，下课清空点赞记录。相关API在LiveRoom中。

```
/**
 * 获取视频点赞监听
 *
 * @return
 */
Observable<LPInteractionAwardModel> getObservableOfAward();

/**
 * 发送点赞请求
 *
 * @param to     被点赞学生的userNumber
 * @param record 全部点赞集合
 */
```

```
14. void requestAward(String to,
    HashMap<String, Integer> record);
```

`LPInteractionAwardModel` 保存了所有历史点赞数据 `model.value.record` 为userId做key，点赞数为value的map，以及本次被点赞的学生的userNumber `model.value.to` 。

16. 点名

点名由老师发起，学生监听并答到。

```
1. routerListener.getLiveRoom().setOnRollCallLis
    OnPhoneRollCallListener() {
2.     @Override
3.     public void onRollCall(int time,
   RollCall rollCallListener) {
4.         // 收到点名
5.
6.         //学生答到API
7.         rollCallListener.call();
8.     }
9.
10.    @Override
11.    public void onRollCallTimeOut() {
12.        // 点名超时
13.    }
14. });
```

可以参考UI实现

17. 外接移动端设备作为摄像头采集

2.5.0 版本开始支持外接设备作为摄像头，仅***主讲人***支持使用，作为摄像头的设备可以通过非参加码的方式进入教室，作为主讲人

的摄像头画面采集,外接设备处理可参考UI实现

- 除链接进教室api外其余方法均在`SpeakQueueVM`,使用`liveRoom.getSpeakQueueVM`获取

获取外接设备进入教室的链接

```
/**
     * 获取扫码视频分享地址
     * @return
     */
    Observable<String>
getObservableOfAsCameraUrl(int
replaceMediaType);
```

链接进教室方式

```
 /**
     * 通过url进教室
     * 使用LiveSDK.enterRoom()
     * @param context
     * @param url
     * @param listener
     * @return
     */
    public static LiveRoom
enterRoom(Context context, String url,
final LPLaunchListener listener);
```

结束投屏和结束投屏回调

```
  /**
     * 结束投屏
     */
```

```
    void stopAsCameraUser();
    /**
     * 结束投屏
     * @return
     */
    Observable<Boolean>
getObservableOfStopAsCamera();
```

其他相关Api

```
/**
     * 是否允许外接设备
     * 主讲人调用
     * @return
     */
    boolean enableAttachPhoneCamera();
    /**
     * 是否有外接设备在推流
     * @return
     */
    boolean hasAsCameraUser();

    /**
     * 被外接设备替换用户
     * @return
     */
    IUserModel getReplacedUser();

    /**
     * 自己是否是被外接设备替换用户
     * @return
     */
    boolean isReplacedUser();

    /**
```

```
     * 是否可以作为外接设备推流
     * 外接设备调用
     * @return
     */
    boolean enableAsCamera();
```

18.LiveRoom其他API

上课/下课

```
liveRoom.requestClassStart();
liveRoom.requestClassEnd();

liveRoom.getObservableOfClassStart().subscrib
liveRoom.getObservableOfClassEnd().subscribe(
```

一般地，requestClassStart和requestClassEnd均由**老师**角色调用

获取当前用户

```
IUserModel currentUser =
liveRoom.getCurrentUser();
```

获取老师用户

```
IUserModel currentUser =
liveRoom.getTeacherUser();
```

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

全体禁言

```
liveRoom.requestForbidAllChat(true);
// 开启全体禁言
liveRoom.requestForbidAllChat(false);
// 关闭全体禁言
Observable<Boolean> getObservableOfForbidAllChatStatus();   // 全体禁言状态KVO
```

单个禁言

单个用户禁言，仅限**老师**角色

```
/**
* 禁言(teacher only)
*
* @param forbidUser 禁言用户
* @param duration   禁言时长
*/
liveRoom.forbidChat(IUserModel forbidUser, long duration);
```

禁言回调(包含其他人被禁言)

```
liveRoom.getObservableOfForbidChat().subscrib
 Consumer<IForbidChatModel>() {
    @Override
    public void accept(IForbidChatModel
iForbidChatModel) {
    }
});
```

当前用户是否被禁言

```
liveRoom.getObservableOfIsSelfChatForbid().su
 Consumer<Boolean>() {
    @Override
    public void accept(Boolean
isChatForbid) {
    }
})
```

自定义事件广播接收

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

设置音频输出

```
LiveSDK.setAudioOutput(LPConstants.VoiceType.
    //通话通道输出
LiveSDK.setAudioOutput(LPConstants.VoiceType.
   //媒体通道输出
```

注意：需要在进教室之前调用

18. 枚举说明

18.1. 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;
    }

    public static LPResolutionType from(int type) {
        switch (type) {
            case 1:
                return HIGH;
            case 2:
                return _720;
            case 3:
                return _1080;
            case 4:
                return _360;
            case 5:
                return _540;
            default:
                return LOW;
        }
    }
}
```

19.出错回调

```
liveRoom.setOnLiveRoomListener(new OnLiveRoomListener() {
    @Override
    public void onError(LPError lpError) {
    }
});
```

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。

错误码

```
public static final int CODE_ERROR_NETWORK_FAILURE = -0x01;//失败、无网
    public static final int CODE_ERROR_NETWORK_MOBILE = -0x02;//当前网络为mobile
    public static final int CODE_ERROR_NETWORK_WIFI = -0x03;//wifi
    public static final int CODE_ERROR_HTTP = -0x04;// 未知错误
    public static final int CODE_ERROR_JSON_PARSE_FAIL = -0x05;// 数据解析失败
    public static final int CODE_ERROR_INVALID_PARAMS = -0x06; // 无效参
```

```
数
7.      public static final int
CODE_ERROR_ROOMSERVER_FAILED = -0x07; //
roomserver登录失败
8.      public static final int
CODE_ERROR_OPEN_AUDIO_RECORD_FAILED =
-0x08;//打开麦克风失败，采集声音失败
9.      public static final int
CODE_ERROR_OPEN_AUDIO_CAMERA_FAILED =
-0x09;//打开摄像头失败，采集图像失败
10.     public static final int
CODE_ERROR_MAX_STUDENT = -0x0A;//人数上限
11.     public static final int
CODE_ERROR_ROOMSERVER_LOSE_CONNECTION =
-0x0B; // roomserver 连接断开
12.     public static final int
CODE_ERROR_LOGIN_CONFLICT = -0x0C; // 被踢下
线
13.     public static final int
CODE_ERROR_PERMISSION_DENY = -0x0D; // 权限
错误
14.     public static final int
CODE_RECONNECT_SUCCESS = -0x0E; // 重连成功
15.     public static final int
CODE_ERROR_STATUS_ERROR = -0x0F; // 状态错误
16.     public static final int
CODE_ERROR_MEDIA_SERVER_CONNECT_FAILED =
-0x10; //音视频服务器连接错误
17.     public static final int
CODE_ERROR_MEDIA_PLAY_FAILED = -0x11; //音视
频播放失败
18.     public static final int
CODE_ERROR_CHATSERVER_LOSE_CONNECTION =
-0x12; // chatserver 连接断开
19.     public static final int
CODE_ERROR_MESSAGE_SEND_FORBID = -0x13; //发
```

```
言被禁止
20.     public static final int
CODE_ERROR_VIDEO_PLAY_EXCEED = -0x14; // 超
出最大播放视频数量
21.     public static final int
CODE_ERROR_LOGIN_KICK_OUT = -0x15; //被踢
22.     public static final int
CODE_ERROR_FORBID_RAISE_HAND = -0x16; //举手
被禁止
23.     public static final int
CODE_ERROR_NEW_SMALL_COURSE = -0x17; // 移动
端禁止新小班课
24.     public static final int
CODE_ERROR_ENTER_ROOM_FORBIDDEN = -0x18; //
禁止进入教室
25.     public static final int
CODE_ERROR_INVALID_SIGNAL_KEY = -0x19; // 错
误的自定义信令类型
26.     public static final int
CODE_ERROR_INVALID_SIGNAL_VALUE = -0x1A; //
自定义信令内容过长
27.     public static final int
CODE_ERROR_SIGNAL_FREQUENCY_TOO_HIGH =
-0x1B; // 自定义信令发送频率过高
28.     public static final int
CODE_ERROR_INVALID_USER_ROLE = -0x1C; //角色
权限错误
29.     public static final int
CODE_ERROR_INVALID_ARGUMENT = -0x1D;// 传入
参数错误;
30.     public static final int
CODE_ERROR_FORBID_AUDIO_DISABLE = -0x1F;//静
音功能被禁止;
31.     public static final int
CODE_ERROR_MIC_OPEN_FORBID = -0x20;//老师禁
止打开麦克风
```

```
32.     public static final int
CODE_WARNING_PLAYER_LAG = -0x21; //直播卡顿
33.     public static final int
CODE_WARNING_PLAYER_MEDIA_SUBSCRIBE_TIME_OUT
 = -0x22; //media subscribe time out
34.     public static final int
CODE_ERROR_WEBRTC_SERVER_DISCONNECTED =
-0x23; // WebRTC音视频服务器断开连接
35.     public static final int
CODE_ERROR_QUESTION_SEND_FORBID = -0x24; //
禁止问答
36.     public static final int
CODE_ERROR_OUTOF_VIDEO_RESOLUTION_LOW =
-0x25; // 超出可设置分辨率范围，已设置最低分辨率
37.     public static final int
CODE_ERROR_OUTOF_VIDEO_RESOLUTION_HIGH =
-0x26; // 超出可设置分辨率范围，已设置最高分辨率
38.     public static final int
CODE_ERROR_LOGIN_UNIQUE_CONFLICT = -0x27;
// 老师进入教室时，教室里面已有老师
39.     public static final int
CODE_ERROR_LOGIN_AUDITION = -0x28;//试听参加
码进入，试听时间过期
40.     public static final int
CODE_ERROR_DEVICE_NOT_SUPPORTED = -0x29;//设
备不支持
41.     public static final int
CODE_ERROR_HOST_UNKNOW = -0x30;//webSocket
host错误
42.     public static final int
CODE_ERROR_MASTERSERVER_LOSE_CONNECTION =
-0x31;//masterserver 断开连接
43.     public static final int
CODE_ERROR_NO_CAMERA_PERMISSION = -0x32; //
没有摄像头权限
```

```
    public static final int CODE_ERROR_EXCEED_MAX_STREAM_NUMBER = -0x33; //超过硬件支持最大路数
    public static final int CODE_ERROR_CLASS_EXPIRED = -0x34; //直播已结束
    public static final int CODE_ERROR_CLASS_NOT_STARTED = -0x35; //课程未不开始不能推流
    public static final int CODE_ERROR_STUDENT_BACKSTAGE = -0x36; //学生在后台
    public static final int CODE_ERROR_STUDENT_UNAVAILABLE = -0x37; //不可用
    public static final int CODE_ERROR_STUDENT_OCCUPIED = -0x38; //被占用
    public static final int CODE_ERROR_SCREEN_SHARE = -0x39; //屏幕分享错误
    public static final int CODE_ERROR_CAMERA_SHARE = -0x3A; //摄像头分享错误
```

20. 混淆规则

```
# 百家云混淆规则
-dontwarn com.baijiahulian.**
-dontwarn com.bjhl.**
-keep class com.baijiahulian.**{*;}
-keep class com.bjhl.**{*;}
-keep class com.baijia.**{*;}
-keep class com.baijiayun.**{*;}
```

```
-keep class org.webrtc.**{*;}
-keep class com.tencent.**{*;}
-keep class org.boom.** {*;}
-keep class org.brtc.**{*;}

# 点播SDK
-keep class tv.danmaku.ijk.**{*;}

# brtc
-keepclassmembers class * {
    @org.brtc.webrtc.CalledByNative
<fields>;
}
-keepclassmembers class * {
    @org.brtc.webrtc.CalledByNative
<methods>;
}

# RxJava混淆规则
-dontwarn sun.misc.**
-keepclassmembers class
rx.internal.util.unsafe.*ArrayQueue*Field*
{
    long producerIndex;
    long consumerIndex;
}
-keepclassmembers class
rx.internal.util.unsafe.BaseLinkedQueueProduc
 {
    rx.internal.util.atomic.LinkedQueueNode
producerNode;
}
-keepclassmembers class
rx.internal.util.unsafe.BaseLinkedQueueConsum
 {
```

```
    rx.internal.util.atomic.LinkedQueueNode consumerNode;
}
-keep class org.apache.log4j.**{*;}
-keep class de.mindpipe.android.logging.log4j.**{*;}
# okhttp3
-dontwarn okhttp3.**
-dontwarn okio.**
-dontwarn javax.annotation.**
-dontwarn org.conscrypt.**
-keepnames class okhttp3.internal.publicsuffix.PublicSuffixDat
# ----------------- kcp ------------------- #
-keep class com.bjy.kcp.** { *; }
-keepclassmembers class com.bjy.kcp.** { *; }
-keepnames class com.bjy.kcp.** { *; }
-keepclassmembernames class com.bjy.kcp.** { *; }
```

集成常见问题

1. 是否支持模拟器

答：直播暂时不支持x86架构，模拟器的话不能使用音视频模块。

2. 视频窗口之间叠加时，显示不出来

答：当视频窗口采用SurfaceView时，会存在SurfaceView叠加的问题，可以采用SurfaceView的setZOrderMediaOverLayer或者setZOrderOnTop来解决，原理请参见官方文档)。

3. 上传带中文名的文件图片崩溃

如果栈中是okhttp检查名字崩溃，则升级到3.11.0，如果更高3.12.x升级到3.12.3以上。

4. 上架google play应用商店被拒，提示tbs 会下载二进制文件

exclude掉sdk依赖的tbs版本，本地依赖
tbs_sdk_noimpl_43799.jar 空实现的jar包。

下载为pdf格式