教室工具包含一系列互动功能，如公告、答题器、测验、问答、点名、点赞、pk。对应的逻辑封装在 `ToolBoxVM` ，通过 `liveRoom.getToolBoxVM()` 获得。

1. 公告

主动获取直播间公告

```
/**
    * 获取公告
    */
    void requestAnnouncement();

    /**
    * 获取公告/分组通知
    *
    * @param group·分组ID
    */
    void requestAnnouncement(int group);
```

设置直播间公告，仅**老师**角色可用

```
 /**
    * changeRoomAnnouncement(0, String announcement, String link)
    * @param announcement
    * @param link
    * @return
    */
```

```
    LPError changeRoomAnnouncement(String announcement, String link);

    /**
     * 老师发送新公告/通知
     *
     * @param group        当前组ID
     * @param announcement 公告/通知正文
     * @param link         点击公告的跳转链接，没有可传空
     * @return null 标识发送成功
     */
    LPError changeRoomAnnouncement(int group, String announcement, String 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();
    }
});
```

2. 测验

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

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);
```

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();
```

3. 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` 。

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

4. 答题器

数据结构说明

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();
```

5. 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
```

```
    */
    void requestAward(String to, HashMap<String, Integer> record);

    /**
     * 发送点赞请求
     * @param to     被点赞学生的userNumber
     * @param key    奖励类型（后端接口返回的key值）key
     * @param record 全部点赞集合
     */
    void requestAward(String to, String key, HashMap<String, LPAwardUserInfo> record);

    /**
     * 批量点赞，不显示动效
     *
     * @param to         被点赞学生的userNumber
     * @param toStudents 被点赞的学生集合，当为to为-1并且toStudents有值时，则不显示动效
     * @param record     全部点赞集合
     * @param key        奖励新类型（后端接口返回的key值）key
     */
    void requestAward(String to, List<String> toStudents, String key, HashMap<String, LPAwardUserInfo> 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` 。

6. 点名

点名由老师发起，学生监听并答到。

```
routerListener.getLiveRoom().setOnRollCallListener(new
 OnPhoneRollCallListener() {
    @Override
    public void onRollCall(int time, RollCall
rollCallListener) {
        // 收到点名

        //学生答到API
        rollCallListener.call();
    }

    @Override
    public void onRollCallTimeOut() {
        // 点名超时
    }
});
```

老师发起点名

```
/**
     * 老师发起点名
     *
     * @param duration 超时时长
     */
    void startRollCall(long duration);

    /**
     * 点名结果回调
     *
     * @return
     */
```

```
13.    Observable<LPRoomRollCallResultModel>
getObservableOfRollCallResult();
14.
15.    /**
16.     * 历史点名结果
17.     *
18.     * @return
19.     */
20.    LPRoomRollCallResultModel
getCacheRollCallResult();
```

`LPRoomRollCallResultModel` 包含答到人员和未答到人员集合

```
1. public class LPRoomRollCallResultModel extends
LPResRoomModel {
2.    /**
3.     * 答到人员
4.     */
5.    @SerializedName("ack_list")
6.    public List<User> ackList;
7.    /**
8.     * 未答到人员
9.     */
10.   @SerializedName("nack_list")
11.   public List<User> nackList;
12. }
```

7. 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();

    /**
```

```
26.     * 是否可以作为外接设备推流
27.     * 外接设备调用
28.     * @return
29.     */
30.    boolean enableAsCamera();
```

下载为pdf格式