

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

```
1. /**
2.   * 获取公告
3.   */
4. void requestAnnouncement();
5.
6. /**
7.   * 获取公告/分组通知
8.   *
9.   * @param group·分组ID
10.  */
11. void requestAnnouncement(int group);
```

设置直播间公告，仅**老师**角色可用

```
1. /**
2.   * changeRoomAnnouncement(0, String
3.   announcement, String link)
4.   * @param announcement
5.   * @param link
6.   * @return
7.   */
```

```

7.   LPErr changeRoomAnnouncement(String
      announcement, String link);
8.
9.   /**
10.    * 老师发送新公告/通知
11.    *
12.    * @param group    当前组ID
13.    * @param announcement 公告/通知正文
14.    * @param link     点击公告的跳转链接，没有可
      传空
15.    * @return null 标识发送成功
16.    */
17.   LPErr changeRoomAnnouncement(int group,
      String announcement, String link);

```

直播间公告变更通知

```

1. liveRoom.getObservableOfAnnouncementChange().
2.   .subscribe(new
      Consumer<IAnnouncementModel>() {
3.     @Override
4.     public void accept(IAnnouncementModel
      iAnnouncementModel) {
5.         String content =
      iAnnouncementModel.getContent();
6.         String url =
      iAnnouncementModel.getLink();
7.     }
8. });

```

2. 测验

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

发布答题

```
1. /**
2.  * 发布答题
3.  *
4.  * @param quizId
5.  * @param forceJoin true: 强制答题 false: 不强制
   答题
6.  */
7. void requestQuizStart(String quizId, boolean
   forceJoin);
```

转发

```
1. /**
2.  * 服务端转发开始答题
3.  * @return
4.  * LPJsonModel:
5.  * {
6.  *   message_type: "quiz_start",
7.  *   quiz_id: {string},
8.  *   force_join: {number} // 0 不强制答题 1 强制答
   题
9.  * }
10. */
11. Observable<LPJsonModel>
   getObservableOfQuizStart();
```

终止答题

```
1. /*
2.  * 终止答题
3.  * @param quizId
```

```
4. */
5. void requestQuizEnd(String quizId);
```

转发终止答题

```
1. /**
2.  * 服务端转发终止答题
3.  * @return
4.  * LPJsonModel:
5.  * {
6.  *   message_type: "quiz_end",
7.  *   quiz_id: {string}
8.  * }
9. */
10. Observable<LPJsonModel>
    getObservableOfQuizEnd();
```

老师公布答案

```
1. /**
2.  * 老师公布答案
3.  *
4.  * @param quizId
5.  * @param solution 后面参数的solution的map都是
   如下形式:
6.  * {
7.  *   "123": 1,// question_id => solution
8.  *   "124": [12, 13],
9.  *   "125": "长江",
10. * }
11. */
12. void requestQuizSolution(String quizId,
    Map<String, Object> solution);
```

服务器转发答案

```
1. /**
2.  * 服务端转发答案
3.  * @return
4.  * LPJsonModel:
5.  * {
6.  *   message_type: "quiz_solution",
7.  *   quiz_id: {string},
8.  *   solution: {
9.  *     "123": 1, // question_id => solution
10.  *     "124": [12, 13],
11.  *     "125": "长江",
12.  *   }
13.  * }
14. */
15. Observable<LPJsonModel>
    getObservableOfQuizSolution();
```

请求当前正在答的题

```
1. /**
2.  * 当前正在答的题
3.  *
4.  * @return
5.  */
6. Observable<LPJsonModel>
    getObservableOfQuizRes();
```

当前正在答的题

```
1. /**
2.  * 当前正在答的题
```

```
3. * @return
4. * LPJsonModel:
5. * {
6. *   message_type: "quiz_res",
7. *   quiz_id: {string}, // 如果当前无答题, 则
   quiz_id为空字符串
8. *   solution: {
9. *     "123": 1, // question_id => solution
10. *     "124": [12, 13],
11. *     "125": "长江",
12. *   }, // 如果曾经提交过, 返回之前提交的结果, 否则
   为空
13. *   force_join: {number}, // 0 不强制答题 1 强制答
   题
14. *   end_flag: {number} // 0 未触发结束答题, 1 已
   触发结束答题
15. * }
16. */
17. Observable<LPJsonModel>
   getObservableOfQuizRes();
```

学生答题

```
1. /**
2. * 学生答题
3. *
4. * @param quizId
5. * @param solution
6. */
7. void submitQuiz(String quizId, Map<String,
   Object> solution);
```

学生答题转发给老师和助教

```
1. /**
2.  * 学生答题发给老师或助教
3.  *
4.  * @param quizId
5.  * @param solution
6.  */
7. void submitQuizToSuper(String quizId,
    Map<String, Object> solution);
```

获取试卷列表(老师)

```
1. /**
2.  * 获取试卷列表
3.  *
4.  * @return
5.  * LPQuizModel中仅有quizId和title
6.  */
7. Observable<List<LPQuizModel>> getListQuiz();
```

新建/更新试卷(老师)

```
1. /**
2.  * 新建/更新试卷
3.  *
4.  * @param lpQuizModel 新建quiz_id和question_id
    和option_id为0
5.  * @return
6.  */
7. LPError saveQuiz(LPQuizModel lpQuizModel);
```

删除试卷(老师)

```
1. /**
2.  * 删除试卷
3.  *
4.  * @param quizId
5.  * @return
6.  */
7. Observable<Boolean> deleteQuiz(long quizId);
```

获取试卷详情/答题详情(老师)

```
1. /**
2.  * 获取试卷详情/答题详情
3.  *
4.  * @param quizId
5.  * @param type 0试卷详情1答题详情
6.  * @return
7.  */
8. Observable<LPQuizModel> getQuizDetail(long
    quizId, LPConstants.LPExamQuizType type);
```

导入试卷(老师)

```
1. /**
2.  * 导入试卷 excel文件
3.  *
4.  * @param excelPath
5.  * @return
6.  */
7. Observable<Boolean> importExcel(String
    excelPath);
```

获取试卷导出地址(老师)


```
1. /**
2.  * 获取试卷导出地址
3.  * @param quizId
4.  * @param type 0 导出试卷 1导出测验结果
5.  * @return
6.  */
7. Observable<String> getExportUrl(long quizId,
    LPConstants.LPExamQuizType type);
```

学生接口获取试卷(学生)

```
1. /**
2.  * 学生接口获取试卷
3.  * @param quizId
4.  * @return
5.  */
6. Observable<LPQuizModel> getQuizInfo(long
    quizId);
```

测验广播列表

```
1. /**
2.  * 测验广播列表
3.  *
4.  * @return
5.  */
6. LPQuizCacheModel getQuizCacheList();
7. /**
8.  * 获取测验广播列表
9.  *
10.  * @return
11.  */
```

```
12. Observable<LPQuizCacheModel>  
    getObservableOfQuizCacheList();
```

大小班小测同步信息请求

```
1. /**  
2.  * 大班课小测同步信息(大班切到小班)  
3. */  
4. void requestRoomQuiz();
```

大小班小测同步信息响应

```
1. /**  
2.  * 大班课小测同步信息  
3.  * @return  
4.  * {  
5.  *   "quiz_id":{string},  
6.  *   "quiz_title":{string}  
7.  * }  
8. */  
9. 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

```
1. /**
```

```
2. * 发送问题
3. *
4. * @param content 提问问题
5. * @return LPErrer
6. * {@link
  LPErrer#CODE_ERROR_INVALID_ARGUMENT} 输入
  内容错误;
7. * {@link
  LPErrer#CODE_ERROR_QUESTION_SEND_FORBID}
  权限错误
8. */
9. LPErrer requestQuestionSend(String content);
10.
11. /**
12. * 获取问答回调
13. */
14. Observable<LPQuestionSendModel>
  getObservableOfQuestionSendRes();
```

获取历史问答

```
1. /**
2. * 请求历史问答
3. *
4. * @return LPErrer {@link
  LPErrer#CODE_ERROR_INVALID_ARGUMENT} 没有
  更多问题信息
5. */
6. LPErrer
  requestQuestionPullReq(LPQuestionPullReqModel
  lpQuestionPullReqModel);
7.
8. /**
9. * 历史问答
```

```
10. */
11. Observable<LPQuestionPullResModel>
    getObservableOfQuestionPullRes();
```

发布、取消、回复问答

```
1. /**
2.  * 发布,取消发布,回复问答
3. */
4. void
    requestQuestionPub(LPQuestionPubTriggerModel
        lpQuestionPubTriggerModel);
5.
6. /**
7.  * 发布,取消发布,回复问答 返回
8. */
9. Observable<LPQuestionPubModel>
    getObservableOfQuestionPub();
```

是否有问答权限

```
1. /**
2.  * 是否有问答权限
3. */
4. 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：

```
1. // 0/选择题 1/判断题
2. public int type;
3. // 答题时长，秒
4. public long duration;
5. // 题目描述
6. private String description;
7. // 答题开始的unix时间戳，即服务端收到
   answer_start_trigger的时间
8. public long timeStart;
9. // 答题完成时是否显示正确答案
10. public boolean isShowAnswer;
11. // 选项
12. 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

```
1. // 选项名，必填
2. public String text;
3. // 是否是正确选项
4. public boolean isRight;
5. //旧版本信令字段，同isRight
6. public boolean isCorrect;
7. // 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
8. public boolean isActive;
9. // 该选项被选择的次数
10. 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：

```
1. // 分组id
2. public int groupId;
3. // user_id
4. public String userId;
5. // user_number
6. public String userNumber;
7. // 人名
8. public String userName;
9. // 排名
10. public int rank;
```

发布答题（触发）

```
1. /**
2.  * 发布答题（触发） 一般由老师调用
3.  *
4.  * @param lpAnswerModel
5.  */
6. void requestAnswerStart(LPAnswerModel
   lpAnswerModel);
```

发布答题（响应）

```
1. /**
2.  * 发布答题（响应）
3.  *
4.  * @return
5.  */
6. Observable<LPAnswerModel>
   getObservableOfAnswerStart();
```

停止/撤销答题（触发）

```
1. /**
2. * 停止/撤销答题（触发）
3. * @param isRevoke 是否是撤销
4. * @param delay 延时结束
5. */
6. void requestAnswerEnd(boolean isRevoke, long
   delay);
```

停止/撤销答题（响应）

```
1. /**
2. * 停止/撤销答题（响应）
3. */
4. Observable<LPAnswerEndModel>
   getObservableOfAnswerEnd();
```

答题数据更新

```
1. /**
2. * 答题数据更新（如有学生答题即可收到更新）
3. */
4. Observable<LPAnswerModel>
   getObservableOfAnswerUpdate();
```

请求历史答题数据（请求）

```
1. /**
2. * 请求历史答题数据（请求）
3. * @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
```

```
4. */
5. void requestAnswerPullReq(String id);
```

请求历史答题数据（返回）

```
1. /**
2. * 请求历史答题数据（返回）
3. * @return map的key为某次答题的id
4. */
5. Observable<Map<Object,
    LPAnswerRecordModel>>
    getObservableOfAnswerPullRes();
```

提交答案

```
1. /**
2. * 提交答案
3. */
4. boolean submitAnswers(LPAnswerModel
    lpAnswerModel);
5.
6. /**
7. * 提交答案
8. * @params timeUsed 本地设置答题用时
9. */
10. boolean submitAnswers(LPAnswerModel
    lpAnswerModel, long timeUsed);
```

请求答题器排名数据

```
1. /**
2. * 请求答题器排名数据
```



```
3. *
4. * @param top 前top名
5. * @param userNumber 自己的userNumber
6. * @param groupId 自己的分组id、id为0表示未分组
   或者是大班身份
7. */
8. void requestAnswerRankList(int top, String
   userNumber, int groupId);
```

答题器排名数据返回

```
1. /**
2. * 答题器排名数据返回
3. */
4. 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。

```
1. /**
2. * 获取视频点赞监听
3. * @return
4. */
5. Observable<LPInteractionAwardModel>
   getObservableOfAward();
6.
7. /**
8. * 发送点赞请求
9. * @param to 被点赞学生的userNumber
10. * @param record 全部点赞集合
```

```

11.    */
12.    void requestAward(String to, HashMap<String,
13.    Integer> record);
14.
15.    /**
16.     * 发送点赞请求
17.     * @param to    被点赞学生的userNumber
18.     * @param key   奖励类型（后端接口返回的key
19.     值）key
20.     * @param record 全部点赞集合
21.     */
22.    void requestAward(String to, String key,
23.    HashMap<String, LPAwardUserInfo> record);
24.
25.    /**
26.     * 批量点赞，不显示动效
27.     *
28.     * @param to    被点赞学生的userNumber
29.     * @param toStudents 被点赞的学生集合，当为to
30.     为-1并且toStudents有值时，则不显示动效
31.     * @param record 全部点赞集合
32.     * @param key   奖励新类型（后端接口返回的
33.     key值）key
34.     */
35.    void requestAward(String to, List<String>
36.    toStudents, String key, HashMap<String,
37.    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。

```
1. routerListener.getLiveRoom().setOnRollCallListener
   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. });
```

老师发起点名

```
1. /**
2.    * 老师发起点名
3.    *
4.    * @param duration 超时时长
5.    */
6. void startRollCall(long duration);
7.
8. /**
9.    * 点名结果回调
10.   *
11.   * @return
12.   */
```

```
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

```
1. /**
2.  * 获取扫码视频分享地址
3.  * @return
4.  */
5.  Observable<String>
   getObservableOfAsCameraUrl(int
   replaceMediaType);
```

链接进教室方式

```
1. /**
2.  * 通过url进教室
3.  * 使用LiveSDK.enterRoom()
4.  * @param context
5.  * @param url
6.  * @param listener
7.  * @return
8.  */
9.  public static LiveRoom enterRoom(Context
   context, String url, final LPLaunchListener
   listener);
```

结束投屏和结束投屏回调

```
1. /**
2.  * 结束投屏
3.  */
4.  void stopAsCameraUser();
```

```
5.  /**
6.   * 结束投屏
7.   * @return
8.   */
9.  Observable<Boolean>
    getObservableOfStopAsCamera();
```

其他相关API

```
1. /**
2.   * 是否允许外接设备
3.   * 主讲人调用
4.   * @return
5.   */
6.  boolean enableAttachPhoneCamera();
7.  /**
8.   * 是否有外接设备在推流
9.   * @return
10.  */
11. boolean hasAsCameraUser();
12.
13. /**
14.   * 被外接设备替换用户
15.   * @return
16.   */
17. IUserModel getReplacedUser();
18.
19. /**
20.   * 自己是否是被外接设备替换用户
21.   * @return
22.   */
23. boolean isReplacedUser();
24.
25. /**
```

```
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