1 功能概述

该模块位于文件夹 PolyvLiveCommonModule/Modules/Interact 下，多个场景都可以共用的一个功能模块，包含了如下互动应用：公告、签到、抽奖、答题卡、和问卷，是对sdk层的`PLVInteractWebview`和js桥`PLVJSBridge`的交互封装，使集成互动功能更简单方便。由`PLVInteractView`作为核心类来集成。

互动模块需要登录 socket，在创建聊天模块的时候，已默认自动登录了 socket，所以集成互动功能，最好先集成聊天模块，如果不需要聊天模块，务必提前做好 socket 的登录。、

2 socket 登录与登出

首先，导入头文件`#import <PLVLiveScenesSDK/PLVSocketManager.h>`，登录代码示例如下：

```
// 获取登录参数
PLVRoomData *roomData = [PLVRoomDataManager sharedManager].roomData;
PLVRoomUser *roomUser = roomData.roomUser;

// 功能配置
// 是否允许使用分房间功能，优先级高于后台的配置，默认为NO-不允许
[PLVSocketManager sharedManager].allowChildRoom = allow;

// Socket 登录管理
PLVSocketUserType userType = [PLVRoomUser sockerUserTypeWithRoomUserType:roomUser.viewerType];
[[PLVSocketManager sharedManager] loginWithChannelId:roomData.channelId viewerId:roomUser.viewerId
viewerName:roomUser.viewerName avatarUrl:roomUser.viewerAvatar actor:nil userType:userType];
```

其次，需要监听 socket 模块的回调，遵循协议 `PLVSocketManagerProtocol` 并添加监听代码示例如下：

```
[[PLVSocketManager sharedManager] addDelegate:self delegateQueue:dispatch_get_main_queue()];
```

代码示例中，`delegateQueue` 参数传入 `dispatch_get_main_queue()` 表示希望回调方法从主线程执行，socket 模块登录成功、失败回调如下：

```
#pragma mark - PLVSocketManager Protocol

/// socket 登录成功回调
- (void)socketMananger_didLoginSuccess:(NSString *)ackString {
    // 可显示socket 登录成功提示
}

/// socket 登录失败回调
- (void)socketMananger_didLoginFailure:(NSError *)error {
    // 可弹出 socket 登录失败弹窗
}
```

最后，离开直播间页面时，需要对 socket 模块进行登出，代码示例如下：

```
[[PLVSocketManager sharedManager] logout];
```

3 核心类介绍

代码如下：

```
PLVInteractView *interactView = [[PLVInteractView alloc] init];
interactView.frame = self.view.bounds;
/// 加载在线 互动页面
[interactView loadOnlineInteract];
/// 显示公告
[interactView openLastBulletin];
```

具体的使用方法请参考`PLVLCCloudClassViewController`、`PLVECWatchRoomViewController`中对`PLVInteractView`接口的调用。

3.1 对外API介绍

`PLVInteractView`定义了如下几个需要在页面中使用的方法：

```
/// 互动视图
///
/// @note 支持 ’答题卡、公告、抽奖、问卷、签到‘；
///       依赖于Socket模块正常运作，若互动视图异常，请先确认Socket已正确连接；
///       添加至相应视图中，并调用加载方法即可；
@interface PLVInteractView : UIView

/// 此时是否不允许转屏（默认NO；接收到不同互动消息时，此值将根据业务要求，相应地变化）
@property (nonatomic, assign, readonly) BOOL forbidRotateNow;

/// 是否保持互动视图在同级视图中最顶层
///
/// @note 互动视图需要最顶层，才能保证接收到最新互动时，可完整地被用户查看
///      (YES:每次互动出现时，自动移至同级最顶层 NO:每次互动出现时，不做层级上的变动；默认为YES)
@property (nonatomic, assign) BOOL keepInteractViewTop;

- (void)openLastBulletin;

#pragma mark - 页面加载
/// 加载在线 互动页面
///
/// @note 为避免自动布局的警告，需在调用此方法前，设置 PLVInteractView 的frame值
- (void)loadOnlineInteract;

/// 加载本地 互动页面
///
/// @param htmlString 本地 html 解析后内容
/// @param baseURL 可访问的文件夹路径（注意是 file:// 开头的 URL）
- (void)loadLocalInteractWithHTMLString:(NSString *)htmlString baseURL:(NSURL *)baseURL;

@end
```

3.2 实现介绍

`PLVInteractView`内部实现了互动应用的逻辑：

3.2.1 初始化

```
- (instancetype)initWithFrame:(CGRect)frame{
    if (self = [super initWithFrame:frame]) {
        /// 初始化数据
        [self setupData];
        /// 初始化UI
        [self setupUI];
        /// 初始化互动应用
        [self setupInteractApps];
    }
    return self;
}
```

- 初始化数据

`setupData`方法中`keepInteractViewTop`是互动视图需要最顶层，才能保证接收到最新互动时，可完整地被用户查看，YES:每次互动出现时，自动移至同级最顶层，NO:每次互动出现时，不做层级上的变动；默认为YES

```
- (void)setupData{
    self.keepInteractViewTop = YES;
}
```

- 初始化UI

`setupUI`方法初始化互动应用Webview`PLVInteractWebview`对象，通过设置`PLVInteractWebview`加载在线、本地资源。

```
- (void)setupUI{
    self.backgroundColor = [UIColor colorWithRed:0.0 green:0.0 blue:0.0 alpha:0.3];
    self.hidden = YES;

    self.interactWebview = [[PLVInteractWebview alloc]init];
    self.interactWebview.delegate = self;

    self.jsBridge.delegate = self;
    //self.jsBridge.debugMode = YES;

    [self.webview addSubview:self.closeBtn];
}
```

- 设置具体的互动应用

`setupInteractApps`设置具体的互动应用，将所需的互动应用添加进来。

3.2.2 互动应用的实现

设置互动应用即将想要的互动应用添加进来。见方法：`setupInteractApps`

```
- (void)setupInteractApps{
    [self.jsBridge addJsFunctionsReceiver:self];
    [self.jsBridge addObserveJsFunctions:@[@"initWebview", @"closeWebview", @"linkClick"]];

    [self addInteractApp:[PLVInteractSignIn class] eventString:PLVSocketInteraction_onSignIn_about];/// 签到
    [self addInteractApp:[PLVInteractBulletin class] eventString:PLVSocketIOChatRoom_BULLETIN_EVENT];/// 公告
    [self addInteractApp:[PLVInteractLottery class] eventString:PLVSocketInteraction_onLottery_about];/// 抽奖
    [self addInteractApp:[PLVInteractAnswer class] eventString:PLVSocketInteraction_onTriviaCard_about];/// 答题
卡
    [self addInteractApp:[PLVInteractQuestionnaire class]
eventString:PLVSocketInteraction_onQuestionnaire_about];/// 问卷
}

- (void)addInteractApp:(Class)interactClass eventString:(NSString *)eventString{
    PLVInteractBaseApp * app = [[interactClass alloc] initWithJsBridge:self.jsBridge];
    app.delegate = self;
    [self.interactDict setObject:app forKey:eventString];
}
```

`setupInteractApps`方法的逻辑分为两步：

1. 实例化具体的互动应用，并设置对应的回调。注意，通用控制不属于业务上的互动应用，他是用于控制所有互动应用的一个通用类，是必须要添加的。
2. 将所有互动应用添加到互动应用webView中。

4 互动应用的实现

4.1 互动应用类简介

互动应用的具体实现在PolyvLiveCommonModule/Modules/Interact目录下与`PLVInteractView`同级目录，共有5个互动应用，每个类分别表示：

- PLVInteractAnswer：答题
- PLVInteractBulletin：公告
- PLVInteractLottery：抽奖
- PLVInteractQuestionnaire：问卷
- PLVInteractSignIn：签到

该目录的其他类：`PLVInteractBaseApp+General`是sdk中`PLVInteractBaseApp`扩展类，该类主要是使用SDK 层`PLVSocketManager`给服务端发送数据。

4.2 具体互动应用实现逻辑

5个互动应用均继承自`PLVInteractBaseApp`，`PLVInteractBaseApp`主要是绑定js桥、发送数据到webView和与代理回调。

5个互动应用子类分别处理各自功能，发组装数据发送到webView。

5 SDK核心类介绍

5.1 PLVInteractBaseApp

PLVInteractBaseApp是互动应用基础类，所有的互动应用以该类作为父类拓展，来实现各自的业务逻辑。

5.1.1 子类重写方法

需要具体的互动应用子类重写的方法。

```
/// 初始化
///
/// @param jsBridge PLVJSBridge对象
- (instancetype)initWithJsBridge:(PLVJSBridge *)jsBridge;

/// 接收互动应用信息
///
/// @param msgString 对象
/// @param jsonDict 对象
- (void)processInteractMessageString:(NSString *)msgString jsonDict:(NSDictionary *)jsonDict;
```

5.1.2 代理回调

```
@protocol PLVInteractBaseAppDelegate <NSObject>
/// 互动应用旋转
- (void)plvInteractAppRequirePortraitScreen:(PLVInteractBaseApp *)interactApp;

/// 互动应用显示隐藏
///
/// @param show YES 显示，NO 隐藏
- (void)plvInteractApp:(PLVInteractBaseApp *)interactApp webviewShow:(BOOL)show;

@end
```

5.1.3 子类使用的方法

父类中定义了一些通用方法可供互动应用子类调用。

```
/// 通知显示旋转
- (void)callRequirePortraitScreen;

/// 通知UI显示
- (void)callWebviewShow;

/// 发送数据到WebView
///
/// @param json 发送数据内容
/// @param event 事件名
- (void)submitResultCallback:(NSString *)json event:(NSString *)event;

/// 发送超时数据到WebView
///
/// @param event 事件名
- (void)submitResultTimeoutCallback:(NSString *)event;
```

5.2 PLVJSBridge

PLVJSBridge是Webview Js交互器，可用于与Webview 进行数据交互)

5.2.1 对外API介绍

```
/// 添加对象作为接收者
///
/// @note 该接收者需要实现对应的Js方法
///
/// @param receiver JS方法回调的接收者
- (void)addJsFunctionsReceiver:(NSObject *)receiver;

/// 添加需要监听的Js方法回调
///
/// @note 需通过 [addJsFunctionsReceiver:] 添加对象作为接收者；该接收者需要实现对应的Js方法；满足以上条件，才能如期收到回调；
///
/// @param jsFunctions 需要监听的Js方法回调数组
- (void)addObserveJsFunctions:(NSArray <NSString *> *)jsFunctions;

#pragma mark - 页面加载
/// 加载在线 url 链接
///
/// @param url url链接
/// @param view 承载 webview 的父视图
- (void)loadWebView:(NSString *)url inView:(UIView *)view;

/// 加载本地 html 文件
///
/// @param htmlString 本地 html 解析后内容
/// @param baseURL 可访问的文件夹路径 (注意是 file:// 开头的 URL)
/// @param view 承载 webview 的父视图
- (void)loadHTMLString:(NSString *)htmlString baseURL:(NSURL *)baseURL inView:(UIView *)view;

/// 加载本地 html 文件
///
/// @note 该方法要求 iOS9 以上；对本地文件读取的兼容性更好
///
/// @param URL html 本地文件路径 (注意是 file:// 开头的 URL)
/// @param readAccessURL 可访问的文件夹路径 (注意是 file:// 开头的 URL)
/// @param view 承载 webview 的父视图
- (void)loadFileURL:(NSURL *)URL allowingReadAccessToURL:(NSURL *)readAccessURL inView:(UIView *)view
API_AVAILABLE(ios(9.0));

#pragma mark - Js交互
/// 向 webview 注入 js 以调用方法
///
/// @param jsFunction js 方法名
/// @param params 需要传递的参数
- (void)call:(NSString *)jsFunction params:(NSArray *)params;
```

方法API调用代码的示例可以在`PLVInteractView`、`PLVInteractBaseApp`和`PLVInteractBaseApp`子类中找到。

5.2.2 代理回调

```
@protocol PLVJSBridgeDelegate <NSObject>

@optional

/// webview 加载成功回调
///
/// @param jsBridge 当前对象本身
- (void)plvJSBridgeWebviewDidFinishLoad:(PLVJSBridge *)jsBridge;

/// webview 加载失败回调
///
/// @param jsBridge 当前对象本身
- (void)plvJSBridgeWebviewDidFailLoad:(PLVJSBridge *)jsBridge withError:(NSError *)error;

/// webview 需要展示或隐藏‘加载指示器’时，将触发此回调
///
/// @note 可通过此回调，来获知合适的时机，进行自定义加载指示器的展示或隐藏；
///       若需自定义加载指示器，请设置 [customActivityIndicator]，设置YES后，内置加载指示器将不显示；
///
/// @param jsBridge 当前对象本身
/// @param loadingShow 是否需要展示或隐藏‘加载指示器’ (YES:需要展示 NO:需要隐藏)
- (void)plvJSBridge:(PLVJSBridge *)jsBridge webviewLodingShow:(BOOL)loadingShow;

/// webview 需要展示确认面板
///
/// @note 当此接收到此回调时，可弹出 UIAlertController 或 一个自定义确认弹窗
///
/// @param jsBridge 当前对象本身
/// @param message 需要展示的信息（可作为 ‘确认弹窗’ 的提示语）
/// @param frame 发起弹窗的页面框架信息
/// @param completionHandler 当确认弹窗被点击后，需回调此Block并附带BOOL参数 (YES:用户选择‘好的’ NO:用户选择‘取消’)
- (void)plvJSBridge:(PLVJSBridge *)jsBridge showConfirmPanelWithMessage:(NSString *)message initiatedByFrame:
(WKFrameInfo *)frame completionHandler:(void (^)(BOOL result))completionHandler;
```

设置代理回调的示例可以在Demo的`PLVInteractView`中找到。