1 功能概述

视频播放是多场景项目中提供的基础功能，带货场景会使用到直播播放器和回放播放器。在带货场景模块中，对直播播放器整个布局区域的元素封装为`PLVECLiveVideoLayout`，对回放播放器整个布局区域的元素封装为`PLVECPlaybackVideoLayout`，它们共同实现`IPLVECVideoLayout`接口。

与云课堂场景不同，带货场景的播放器布局不包括控制栏，控制栏是通过叠加在播放器上层的Fragment来实现的，如回放的`PLVECPalybackHomeFragment`，具体细节见*控制栏*章节

2 使用演示

以下示例了如何创建直播播放器布局、回放播放器布局，以及它们的初始化及视频播放，代码如下：

```
// PLVECLiveEcommerceActivity.java

if (liveRoomDataManager.getConfig().isLive()) {
    // 页面底层播放器 - 直播
    videoLayout = new PLVECLiveVideoLayout(this);
} else {
    // 页面底层播放器 - 回放
    videoLayout = new PLVECPlaybackVideoLayout(this);
}
// 播放器容器
FrameLayout videoContainer = findViewById(R.id.plvec_fl_video_container);
// 添加播放器到容器中
videoContainer.addView((View) videoLayout, FrameLayout.LayoutParams.MATCH_PARENT,
FrameLayout.LayoutParams.MATCH_PARENT);
// 初始化播放器、播放
videoLayout.init(liveRoomDataManager);
videoLayout.setOnViewActionListener(new IPLVECVideoLayout.OnViewActionListener() {
    @Override
    public void onCloseFloatingAction() {
        if (floatingWindowBinder != null) {
            floatingWindowBinder.dismiss();
        }
        //主动关闭浮窗时，播放器静音
        videoLayout.setPlayerVolume(0);
        isUserCloseFloatingWindow = true;
    }
});
//当前activity 可以手势操作暂停和播放
initGesture();
videoLayout.startPlay();
```

上述方法的具体用例可以在polyv demo项目中的`PLVECLiveEcommerceActivity`找到。

3 接口介绍

3.1 IPLVECVideoLayout

带货场景下，针对播放器布局进行封装的接口。定义了：

1、外部直接调用的方法（包括common通用方法、live直播特有方法、playback回放特有方法） 2、需要外部响应的事件监听器

具体各个方法的用途、调用参数及返回值等，请参考`IPLVECVideoLayout`接口定义。

4 实现介绍

4.1 PLVECLiveVideoLayout

带货场景下的直播播放器布局，实现`IPLVECVideoLayout`接口。

该布局包含的元素有：直播播放器、子播放器、Logo、屏幕中央的播放暂停按钮等UI。

下面会列举介绍该布局中涉及到的主要方法。

4.1.1 初始化view方法

`PLVECLiveVideoLayout` 继承于 `FrameLayout`，在构造器中对使用 `initView` 方法对 `view` 进行了初始化处理。

```
public PLVECLiveVideoLayout(@NonNull Context context, @Nullable AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    initView();
}

private void initView() {
    // 布局初始化
    LayoutInflater.from(getContext()).inflate(R.layout.plvec_live_player_layout, this, true);

    // findView....

    // 其它初始化方法
    initVideoView();
    initSubVideoViewChangeListener();
}
```

4.1.2 初始化数据方法

`PLVECLiveVideoLayout` 的 `init` 方法是对外API，需要外部传入 `IPLVLiveRoomDataManager` 后进行初始化。

```
// 播放器presenter
private IPLVLivePlayerContract.ILivePlayerPresenter livePlayerPresenter;

public void init(IPLVLiveRoomDataManager liveRoomDataManager) {
    this.liveRoomDataManager = liveRoomDataManager;

    livePlayerPresenter = new PLVLivePlayerPresenter(liveRoomDataManager);
    livePlayerPresenter.registerView(livePlayerView);
    livePlayerPresenter.init();
    livePlayerPresenter.setAllowOpenAdHead(isAllowOpenAdHead);
}

// 创建播放器mvp-view
private IPLVLivePlayerContract.ILivePlayerView livePlayerView = new PLVAbsLivePlayerView() {
     //...
}
```

代码遵循 `MVP` 模式，一切业务逻辑都是通过 `Presenter` 来进行操作，隔离了视图层 `View` 和数据层 `Model`。

4.1.3 开始播放方法

`PLVECLiveVideoLayout` 的 `startPlay` 方法是对外API，该方法内部会调用 `PLVLivePlayerPresenter` 的 `startPlay` 方法来进行视频的加载及播放。

```
@Override
public void startPlay() {
    livePlayerPresenter.startPlay();
}
```

`PLVECLiveVideoLayout` 内部持有 `PLVLivePlayerPresenter`，因此除了内部可以自由调用播放器相关的方法外，也能将其封装成接口提供给外层调用，`startPlay` 方法就是其中一个例子。

4.2 PLVECPlaybackVideoLayout

带货场景下的回放播放器布局，实现 `IPLVECVideoLayout` 接口

该布局包含的元素有：回放播放器、子播放器、Logo、屏幕中央的播放暂停按钮等UI。

下面会列举介绍该布局中涉及到的主要方法。

4.2.1 初始化view方法

`PLVECPlaybackVideoLayout` 继承于 `FrameLayout`，在构造器中对使用 `initView` 方法对 `view` 进行了初始化处理。

```
public PLVECPlaybackVideoLayout(@NonNull Context context, @Nullable AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    initView();
}

private void initView() {
    // 布局初始化
    LayoutInflater.from(getContext()).inflate(R.layout.plvec_playback_player_layout, this, true);

    // findView....

    // 其它初始化方法
    initSubVideoViewChangeListener();
}
```

4.2.2 初始化数据方法

`PLVECPlaybackVideoLayout` 的 `init` 方法是对外API，需要外部传入 `IPLVLiveRoomDataManager` 后进行初始化。

```
// 播放器presenter
private IPLVPlaybackPlayerContract.IPlaybackPlayerPresenter playbackPlayerPresenter;

@Override
public void init(IPLVLiveRoomDataManager liveRoomDataManager) {
    this.liveRoomDataManager = liveRoomDataManager;

    playbackPlayerPresenter = new PLVPlaybackPlayerPresenter(liveRoomDataManager);
    playbackPlayerPresenter.registerView(playbackPlayerView);
    playbackPlayerPresenter.init();
    playbackPlayerPresenter.setAllowOpenAdHead(isAllowOpenAdhead);
}

// 创建播放器mvp-view
private IPLVPlaybackPlayerContract.IPlaybackPlayerView playbackPlayerView = new PLVAbsPlaybackPlayerView() {
    //...
}
```

4.2.3 开始播放方法

`PLVECPlaybackVideoLayout` 的 `startPlay` 方法是对外API，该方法内部会调用 `PLVPlaybackPlayerPresenter` 的 `startPlay` 方法来进行视频的加载及播放。

```
@Override
public void startPlay() {
    playbackPlayerPresenter.startPlay();
}
```

`PLVECPlaybackVideoLayout` 内部持有 `PLVPlaybackPlayerPresenter`，因此除了内部可以自由调用播放器相关的方法外，也能将其封装成接口提供给外层调用， `startPlay` 方法就是其中一个例子。

5 控制栏

带货场景的直播播放器布局 `PLVECLiveVideoLayout` 和回放播放器布局 `PLVECPlaybackVideoLayout` 均不包括控制栏的相关控件，控制栏是通过叠加在播放器布局上层的Fragment实现的

5.1 回放控制栏 - PLVECPalybackHomeFragment

`PLVECPalybackHomeFragment` 是回放首页页面，包括主持人信息、播放控制、进度条、更多

其中控制播放的相关操作是通过调用 `PLVECLiveEcommerceActivity` 设置的回调接口 `playbackHomeViewActionListener` 实现的，下面以切换暂停播放的按钮举例：

```
// PLVECPalybackHomeFragment.java
@Override
public void onClick(View v) {
    int id = v.getId();
    if (id == R.id.play_control_iv) {
        // 这里调用了回调接口的onPauseOrResumeClick方法切换暂停或播放状态
        if (onViewActionListener != null) {
            v.setSelected(onViewActionListener.onPauseOrResumeClick(v));
        }
    }
    // 其它按钮的点击事件...
}
```

```
// PLVECLiveEcommerceActivity.java
private PLVECPalybackHomeFragment.OnViewActionListener playbackHomeViewActionListener = new
PLVECPalybackHomeFragment.OnViewActionListener() {
    @Override
    public boolean onPauseOrResumeClick(View view) {
        // 接口实现中通过调用videoLayout，即播放器布局的方法，来实现播放控制
        if (!videoLayout.isSubVideoViewShow()) {
            if (videoLayout.isPlaying()) {
                videoPause();
                return false;
            } else {
                videoResume();
                return true;
            }
        }
        return false;
    }
    // 其它接口方法实现...
}
```