移动互联网技术及应用

大作业报告

姓名	班级	学号
曾旺丁	**2017211311**	**2017211520**

2020.6

目录

1. 相关技术

1，视频播放页整体采用 MediaPlayer 和 RecyclerView 实现
2，MediaPlyaer 可以结合 SurfaceView 自定义视频播放页的细节
3，RecyclerView 可以实现滑动切换视频播放页
4，点赞，喜欢等图标则通过 ImageView 实现
5，滑动条则使用 SeekBar 实现，可以观看当前进度，也可以调整进度
6，缓冲加载动画使用开源的自定义 View 实现

2. 系统功能需求

本次课程大作业为视频播放器 APP 的开发，需要开发一个视频播放器，从指定的 URL 获取视频资源，然后进行播放。在满足基础要求之后，可以新增视频播放相关的功能。

本项目可以个人完成，也可以组队完成。我们采用组队的方式完成此次大作业，我们将项目划分为显示视频列表与视频播放两个基本功能模块，分别进行开发，最后再进行系统组装与集成。

此次作业中，我负责的是视频播放功能模块的设计与编码。整体而言，我采用了一个类似抖音的设计风格，除基础的视频播放功能外，还具有滑动视频切换，点赞，进度显示与调整等功能。播放页的总体布如下：

3. 系统设计与实现

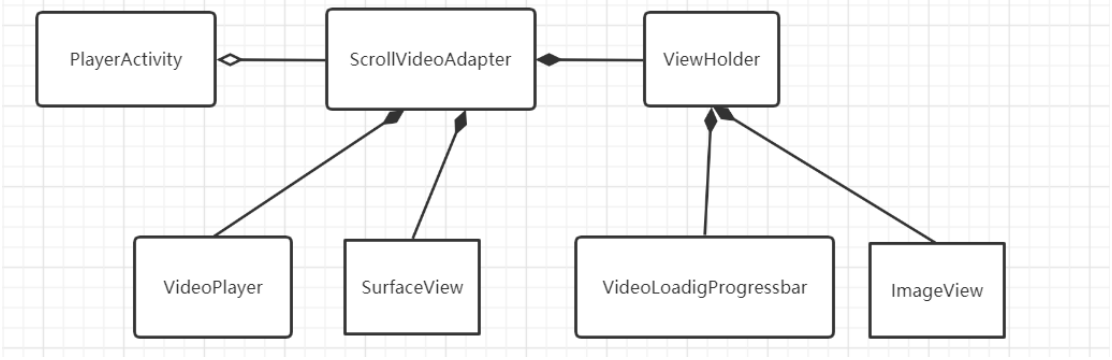
3.1 总体设计

视频播放页为一个 Activity，采用 MediaPlayer 配合 SurfaceView 进行视频播放，使用 RecyclerView 进行滑动页面的切换。

3.2 系统组成

此播放器使用面向对象的编程方式实现，所有模块均设计成类，系统的主要组成类如下：

PlayerActivity
ScrollVideoAdapter
VideoPlayer
VideoLoadigProgressbar



它们的关系大致如图所示（为简洁，还有一些系统自带的 View 类并未标出）。

3.3 模块设计

PlayerActivity:视频播放的 Activity

ScrollVideoAdapter： RecyclerView 的适配器，用于滑动切换视频

VideoPlayer： 通过扩展 MediaPlayer 实现的播放器，具有暂停，播放，获取播放进度，设置播放进度，播放进度刷新等功能。

VideoLoadigProgressbar： 自定义 View,用于在视频加载时显示缓冲动画。

3.4 关键代码

视频播放部分的关键代码在类 ScrollVideoAdapter 和 VideoPlayer，前者用于实现滑动页面，后者用于播放视频。这里给出 ScollVideoAdapter 的关键部分代码注释，对于次要方法，只给出方法原型：

```
1.  public class ScrollVideoAdapter extends VideoPlayAdapter<ScrollVideoAdapter.
    ViewHolder> {
2.      private Context mContext;
3.      private int mCurrentPosition;
4.      // 当前显示的 ViewHolder
5.      private ViewHolder mCurrentHolder;
6.      // 视频播放器
7.      private VideoPlayer videoPlayer;
8.      // 显示视频的视图
9.      private TextureView textureView;
10.     // 视频列表
11.     private ArrayList<Message> videoList;
12.     private Handler handler;
13.     public ScrollVideoAdapter(Context mContext, final ArrayList<Message> vid
    eoList) {
14.     }
15.
16.     @NonNull
17.     @Override
18.     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int view
    Type) {
19.
20.     }
21.
22.     // 滑动开始，下一页面可见时调用，在此处加载封面
23.     @Override
24.     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

25.         RequestOptions options = new RequestOptions().diskCacheStrategy(Disk
    CacheStrategy.RESOURCE);
26.         // 加载封面
27.         Glide.with(mContext).load(videoList.get(position).getAvactar()).appl
    y(options).into(holder.ivCover);
```

```
28.     }
29.
30.     @Override
31.     public int getItemCount() {
32.         return videoList.size();
33.     }
34.
35.     // 滑动完成时，选中一个新页面时调用
36.     @Override
37.     public void onPageSelected(int itemPosition, View itemView) {
38.         mCurrentPosition = itemPosition;
39.         mCurrentHolder = new ViewHolder(itemView);
40.         // 开始播放当前页对应的视频
41.         playVideo(videoList.get(itemPosition).getFeedurl());
42.     }
43.
44.     // 开始播放位于 url 处的视频
45.     private void playVideo(String url) {
46.         videoPlayer.reset();
47.         // 显示缓冲动画视图
48.         mCurrentHolder.pbLoading.setVisibility(View.VISIBLE);
49.         // 设置视频播放器的回调接口，策略与实现分离，VideoPlayer 暴露一个接口处理
    自身的一些事件
50.         videoPlayer.setOnStateChangeListener(new VideoPlayer.OnStateChangeLi
    stener() {
51.             @Override
52.             public void onReset() {
53.                 mCurrentHolder.ivCover.setVisibility(View.VISIBLE);
54.                 mCurrentHolder.pbLoading.setVisibility(View.INVISIBLE);
55.             }
56.             // 开始渲染时调用
57.             @Override
58.             public void onRenderingStart() {
59.                 mCurrentHolder.ivCover.setVisibility(View.GONE);
60.                 mCurrentHolder.pbLoading.setVisibility(View.INVISIBLE);
61.             }
62.
63.             // 进度更新时调用
64.             @Override
65.             public void onProgressUpdate(float per) {
66.                 mCurrentHolder.pg_bar.setProgress((int)(per * 100));
67.             }
68.
69.             // 暂停时调用
```

```
            @Override
            public void onPause() {
                mCurrentHolder.pbLoading.setVisibility(View.INVISIBLE);
                mCurrentHolder.pg_bar.setVisibility(View.VISIBLE);
            }

            // 播放停止时调用
            @Override
            public void onStop() {
                mCurrentHolder.ivCover.setVisibility(View.VISIBLE);
                mCurrentHolder.pbLoading.setVisibility(View.INVISIBLE);
            }

            // 播放完成时调用
            @Override
            public void onComplete() {
                videoPlayer.start();
            }
        });
        if (textureView.getParent() != mCurrentHolder.flVideo) {
            if (textureView.getParent() != null) {
                ((FrameLayout) textureView.getParent()).removeView(textureVi
ew);
            }
            mCurrentHolder.flVideo.addView(textureView);
        }
        videoPlayer.setDataSource(url);
        videoPlayer.prepare();
    }
  // 滑动释放时调用
    public void release() {
         videoPlayer.release();
     }

     class ViewHolder extends RecyclerView.ViewHolder{
         private FrameLayout flVideo;
         private ImageView ivCover;
         private VideoLoadingProgressbar pbLoading;
         private ImageView im_like;
         private ImageView im_favor;
         private ImageView im_star;
         private TextView tv_like;
         private TextView tv_favor;
         private TextView tv_star;
```

```
113.
114.     }
115. }
```

VideoPlayer 是一个我利用 MediaPlayer 实现的播放器，它暴露了一个 OnStateChangeListener 类给外部类，用以实现内部事件的处理。同时，它本身具有多种方法，可以实现视频的暂停，进度设置等等。在此，我给出它的关键代码，对一些次要方法，仅给出原型：

```
1.  public class VideoPlayer {
2.      // 内嵌 mediaPlayer
3.      private MediaPlayer mediaPlayer;
4.      // 播放器内部状态
5.      private State state = State.IDLE;
6.      // 状态改变监听器，暴露给其它类的接口
7.      private OnStateChangeListener onStateChangeListener;
8.      // 用于发布延时任务，实现进度条实时更新
9.      private Handler handler;
10.     private boolean prePause;
11.
12.     // 构造函数
13.     public VideoPlayer() {
14.
15.     }
16.
17.     // 绑定 TextureView
18.     public void setTextureView(TextureView textureView) {
19.
20.     }
21.
22.     // 重置
23.     public void reset() {
24.
25.     }
26.
27.     // 设置视频 url
28.     public void setDataSource(String url) {
29.
30.     }
31.
32.     // 加载视频，准备播放
33.     public void prepare() {
34.
35.     }
36.
```

```
37.     // 开始播放
38.     public void start() {
39.
40.     }
41.
42.     // 设置进度
43.     public void seekTo(int progress){
44.         mediaPlayer.seekTo((int)(progress  * getDuration() / 100));
45.     }
46.
47.     // 刷新进度条
48.     private void refreshProgress() {
49.         if (state != State.PLAYING) {
50.             return;
51.         }
52.         // handler 一个延时任务，继续调用 refreshProgress,实现重复刷新
53.         handler.postDelayed(new Runnable() {
54.             @Override
55.             public void run() {
56.                 if (mediaPlayer == null||state!=State.PLAYING) {
57.                     return;
58.                 }
59.                 if (onStateChangeListener != null) {
60.                     onStateChangeListener.onProgressUpdate(mediaPlayer.getCu
    rrentPosition() * 1f / mediaPlayer.getDuration());
61.                 }
62.                 refreshProgress();
63.             }
64.         }, 100);
65.     }
66.
67.     // 播放暂停
68.     public void pause() {
69.
70.     }
71.
72.     // 停止播放
73.     public void stop() {
74.
75.     }
76.
77.     public void release() {
78.
79.     }
```

```
80.
81.     // 播放器当期状态
82.     public State getState() {
83.         return state;
84.     }
85.
86.     // 视频时长
87.     public int getDuration() {
88.         return mediaPlayer.getDuration();
89.     }
90.
91.     // 设置事件监听器
92.     public void setOnStateChangeListener(OnStateChangeListener onStateChange
    Listener) {
93.         this.onStateChangeListener = onStateChangeListener;
94.     }
95.
96.     // 暴露给其它类的事件处理接口
97.     public interface OnStateChangeListener {
98.         void onReset();
99.
100.         void onRenderingStart();
101.
102.         void onProgressUpdate(float per);
103.
104.         void onPause();
105.
106.         void onStop();
107.
108.         void onComplete();
109.     }
110.
111.     // 状态枚举
112.     public enum State {
113.         // 空闲
114.         IDLE,
115.         // 准备中
116.         PREPAREING,
117.         // 播放中
118.         PLAYING,
119.         // 暂停
120.         PAUSE,
121.         // 播放停止
122.         STOP,
```

```
123.         // 播放结束
124.         COMPLETE
125.     }
126. }
```

以上两个类实现了系统播放视频部分的核心逻辑，其它的类只是 Android 自带的视图类或其它第三方开源的类库，在此不多赘述。值得一提的是，我使用的 RecyclerView 并非官方自带的 RecyclerView，而是一个开源的第三方库，它对滑动视频切换提供了更好的支持。

4. 系统可能的扩展

本播放器只是一个具备基础功能的播放器，可以再次之上进行进一步的开发。包括但不限于完善已有功能，增加新的功能。

功能完善：

点赞时，可以增加一个动态的动画，这可以使得用户体验更加流畅而不显生硬，不过由于开发成本的限制，我并未在此版本添加该功能。

功能新增：

可以增加评论功能。

用户可以上传自己的作品。

增加社交功能，在此基础之上增加关注，私信等功能。

5. 总结体会

移动互连网的存在使移动终端具有了多种可能，没有移动互联网的发展，不可能有如今移动端数不胜数的 App。从 GSM，到 GPRS,再到 3G,4G，我们把手机从一个拨打电话的工具变成了如今生活中具有多种功能的移动终端设备，这得益于移动互联网的发展和技术的进步。

移动互联网即将进入 5G 时代，移动端的开发又会有一番新的面貌，这种进步甚至于会改变人们的生活习惯。比如从 4G 开始，逐渐繁荣的移动支付，网络直播等新型移动端产品，已分别成为生活消费和业余娱乐的重要组成部分。想来，这种打破传统的新型产品在 5G 时代还会出现。

本学期的移动互联网技术课程分为两个部分，一为理论讲解，主要介绍移动通信网的组成；二为开发实践，主要介绍移动应用的开发技术。移动通信网带来了可移动的高速信息通路，为后者移动应用的存在奠定了基础；而移动应用的需求又间接推动了移动通信网的进步。

本课程既有理论也有实践，总体而言，是一次愉快的上课体验，对此，需感谢各位老师的辛勤付出！