

猪猪视频

一个安卓视频播放器

详细介绍

该软件主要分为多个板块，登录注册页面，视频播放页面，个人中心页面等。

登录注册页面

1. 用户信息存储于SQLite数据库中，在进行登录验证时，从数据库中读取数据进行判断，注册同理（保证用户id跟用户名唯一）。

自己封装实现了DbcUtils类以及DbCointect类用来连接以及操作数据库。

以下是封装实现的该应用的一些数据库操作。

```
新用户创建 插入数据
Params: helper
        user

1 usage  ↳ ydy567
public static void insert(SQLiteOpenHelper helper, User user) {...}

获取用户信息, 进行判断
5 usages  ↳ ydy567
public static User query(SQLiteOpenHelper helper, String username) {...}

更新用户密码
Params: helper
        username
        newPassword

1 usage  ↳ ydy567
public static void updatePassword(SQLiteOpenHelper helper, String username, String newPassword) {...}

插入视频数据
Params: helper

no usages  ↳ ydy567
public static void insertVideo(SQLiteOpenHelper helper, Video video) {...}
```

2. 实现了记住密码功能，当用户登录成功时点击保存密码，那么下次用户打开APP或者是进入到登录页面时，就会进行自动填充，不需要再次输入。

使用SharedPreferences类来将记住密码的账号密码存储在缓存中，如果下一个不选择记住密码则进行清空。初始为空值。

```
// 用来记录密码缓存
SharedPreferences spf = getSharedPreferences( name: "users", MODE_PRIVATE);
name = spf.getString( s: "loginname", s1: "");
password = spf.getString( s: "loginpassword", s1: "");
isSave = spf.getBoolean( s: "issave", b: false);
```

视频播放页面

ListView -- ListViewFragment

使用简单的继承自BaseFragment的实现类，其显示内容为数据库中所有用户的所有视频，通过查询数据库的方式去InitData，单个item实现点击播放以及暂停功能。

RecycleView -- ChannelFragment

抖音 -- TikTokListFragment

同样继承自BaseFragment，但是这里的显示内容为爬虫所获取到的存储在云端的视频连接进行显示。

```
public class TikTokListFragment extends BaseFragment
```

在上面类中实现View视图以及适配器

布局设置

```
mRecyclerView.setLayoutManager(new GridLayoutManager(getContext(), spanCount: 2));
```

每一行使用两列布局，实现两个视频同时在一列显示，扩充视图中的视频容量。

视图显示

fragment_tiktok_list 设置主要格式，里面就设置一个RecycleView，然后定义了一个item_tiktok_list来设置每一块的布局，包括视频跟标题。

适配器设置

```
▲ ydy567
@NonNull
@Override
public TikTokListViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {..}

▲ ydy567
@Override
public void onBindViewHolder(@NonNull TikTokListViewHolder holder, int position) {...}

▲ ydy567
@Override
public int getItemCount() { return data == null ? 0 : data.size(); }

1 usage ▲ ydy567
public void setImpl(int id) { mId = id; }
```

onCreateViewHolder方法用于创建并返回一个TikTokListViewHolder对象。这个对象表示RecycleView中每个列表项的视图。

onBindViewHolder方法用于将数据绑定到TikTokListViewHolder中的视图上。它从数据集合中获取特定位置的TiktokBean对象，然后将其标题设置到TextView上，将封面图加载到ImageView中。

```
4 usages  ↲ ydy567 *
public class TikTokListViewHolder extends RecyclerView.ViewHolder {

    3 usages
    public ImageView mThumb;
    2 usages
    public TextView mTitle;
    2 usages
    public int mPosition;

    1 usage  ↲ ydy567 *
    public TikTokListViewHolder(@NonNull View itemView) {
        super(itemView);
        mThumb = itemView.findViewById(R.id.iv_thumb);
        mTitle = itemView.findViewById(R.id.tv_title);

        ↲ ydy567 *
        itemView.setOnClickListener(new View.OnClickListener() {
            ↲ ydy567 *
            @Override
            public void onClick(View v) {
                TikTok2Activity.start(itemView.getContext(), mPosition);
            }
        });
    }
}
```

TikTokListViewHolder 是一个内部类，表示RecyclerView中每个列表项的视图持有者。它包含了列表项中显示的ImageView和TextView，以及一个点击事件监听器。当列表项被点击时，它会启动TikTok2Activity，并传递相应的位置信息。

个人中心页面

这里主要实现一些APP的关于账户的操作，包括退出登录，修改密码，显示个人信息等操作。

该页面通过Intent进行组件间的通信，从登录页面获取到当前登录用户的信息，并显示在用户名中。



用户名: user_1000



关于我们



密码修改



退出登入

修改密码功能，为了减少Fragment的设计数量，选择使用弹窗的形式进行操作。



通过 `LayoutInflater` 创建一个自定义的视图 (`R.layout.dialog_update_password`)，该视图包含三个输入框用于输入原密码、新密码和确认新密码。在点击确定按钮时，可以通过 `view.findViewById()` 获取输入框的引用，并获取输入的内容进行相应的处理。确保在你的布局文件中包含了这三个输入框。

我的页面

该页面功能较为简单，用来显示用户个人所上传的视频信息。主要操作在于将 LoginActivity 中的登录信息传递到该 Fragment。

(上面个人中心页面也使用该方式) 首先是使用 Intent 通信，将 LoginActivity 中的登录信息，包括个人账号名以及密码 (用于修改密码页面作用) 等信息传递到 MainActivity。然后在这个里面创建加入 Fragment 的时候，使用 `setArguments` 方法，将从 Login 接收到的消息传递给 Fragment 组件。然后通过 Adapter 参数构造传递给数据构造 `InitData` 函数，然后再去数据库查询相关用户的视频信息。

仿Youtube 视频播放页面

效果展示：



主要布局 XML 说明如下：

首先在 `fragment_channel` 页面显示视频列表，该列表最外层使用 `SwipeRefreshLayout` 布局，这是 Google 官方推荐的下拉刷新布局控件，只需要把 `RecyclerView` 或者 `ListView` 放在里面就可以实现简单的下拉刷新。

layout

- activity_custom_ijk_player.xml
- activity_danmaku_player.xml
- activity_detail.xml
- activity_hello.xml
- activity_layout_common.xml
- activity_login.xml
- activity_main.xml
- activity_pad.xml
- activity_register.xml
- activity_tiktok2.xml
- activity_youtube_video_play.xml 视频播放页面
- dialog_update_password.xml
- fragment_channel.xml 频道视频列表
- fragment_extension.xml
- fragment_list.xml
- fragment_list_view.xml
- fragment_live.xml
- fragment_pip.xml
- fragment_play_list.xml
- fragment_recycler_view.xml
- fragment_tiktok_list.xml
- item_tik_tok.xml
- item_tiktok_list.xml
- item_video.xml
- layout_ad_control_view.xml
- layout_definition_control_view.xml
- layout_float_controller.xml
- layout_rate_item.xml
- layout_rate_pop.xml
- layout_tiktok_controller.xml
- layout_youtube_comment.xml 视频评论项布局
- layout_youtube_post.xml 单条视频项信息布局

下拉刷新SwipeRefreshLayout

在需要刷新视频内容的ChannelFragment中，代码如下。

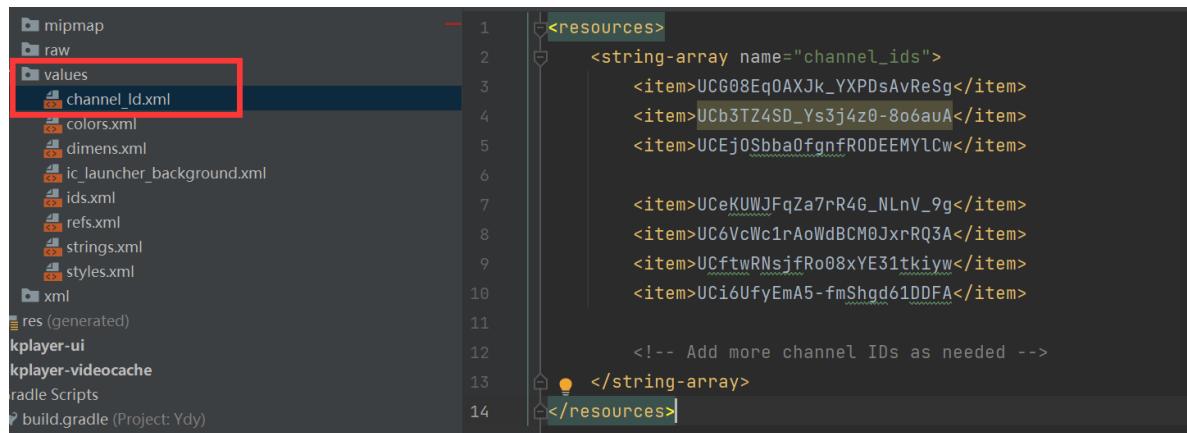
其中，刷新算法update channelId通过一个 `List<String> remainingChannels`，每次从中选择随机项（视频id），然后从列表中移除。当移除为空的时候，使用 `Collections.shuffle(remainingChannels);` 进行打乱，从而进行下一轮的随机刷新选择。

```
public class ChannelFragment extends Fragment {
    private SwipeRefreshLayout swipeRefreshLayout;
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
        swipeRefreshLayout = view.findViewById(R.id.swipeRefreshLayout);
        // 下拉刷新时的监听事件
        swipeRefreshLayout.setOnRefreshListener(this::refreshContent);
    }
}
```

```
/** 下拉刷新：随机获取 channelId -> 清空数据 -> 网络请求 */
2个用法
private void refreshContent() {
    update channelId();
    _data.clear();
    new RequestYoutubeAPI().execute();
}

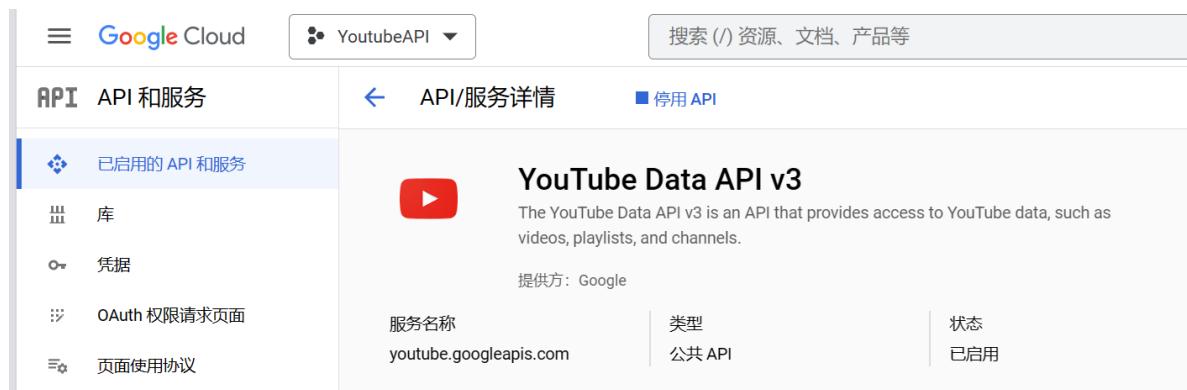
/** 从资源文件中获取随机channel Id */
1个用法
private void update channelId() {
    if (remainingChannels.isEmpty()) {
        for (String channelId : _channelIds) {
            remainingChannels.add(channelId);
        }
        Collections.shuffle(remainingChannels);
    }
    CHANNEL_ID = remainingChannels.remove(0);
}
```

在values/channel_id.xml下，提供了几个Youtube频道爬取id：



Googleapis爬取视频

由于是爬取的Youtube视频，需要到Google的开发者控制台<https://console.cloud.google.com/> 下申请启用`YouTube Data API v3` 服务。在“凭据”处获取API KEY，此密钥用于访问爬取服务googleapis。



The screenshot shows the Google Cloud Platform API library interface. On the left, there's a sidebar with 'API' selected, followed by 'API 和服务'. Under '已启用的 API 和服务', it lists '库', '凭据', 'OAuth 权限请求页面', and '页面使用协议'. On the right, the main panel displays the 'YouTube Data API v3' service details. It includes a red play button icon, a brief description: 'The YouTube Data API v3 is an API that provides access to YouTube data, such as videos, playlists, and channels.', the provider 'Google', and service details: '服务名称: youtube.googleapis.com', '类型: 公共 API', and '状态: 已启用'.

```
/** * @return 返回 googleapis URL, 爬取频道视频Json信息 */  
2个用法  
private static String CHANNEL_GET_URL()  
{  
    return "https://www.googleapis.com/youtube/v3/search?part=snippet&order=date&channelId="  
        + CHANNEL_ID + "&maxResults=20&key=" + GOOGLE_YOUTUBE_API_KEY;  
}
```

在其中，`maxResults`表示最大请求数量，比如如果是访问频道视频，最多会爬取20个视频信息。`googleapis`获取的频道链接内容是Json格式，需要做Parse解析工作，得到需要的视频信息列表。

OkHTTP 请求Url + AsyncTask 异步网络请求

上面准备工作完成后，需要对URL进行请求，并处理响应(response)，通过响应的内容 (Json字符串) 进行解析，得到数据。在使用OkHTTP的时候，在ChannelFragment中，进行引用：

```
import okhttp3.OkHttpClient;  
import okhttp3.Request;  
import okhttp3.Response;
```

`RequestYoutubeAPI`类是`ChannelFragment`中的内部类，用于`GoogleapiURL`的请求。它是一个继承自`AsyncTask`的类。由于主线程用于处理 UI 事件，如果在主线程中执行一些耗时的操作（例如网络请求、文件读写等），可能会导致界面卡顿或 ANR (Application Not Responding) 错误。

为了解决这个问题，可以使用`AsyncTask`将耗时的任务放在后台线程（子线程）中执行，而在主线程（UI线程）中执行一些与 UI 相关的操作。在早期版本的 Android 中，`AsyncTask`主要用于在后台线程执行异步任务，并在主线程中更新 UI。它不是传统意义上的多线程或协程。它是一种轻量级的异步任务框架，其内部实现依赖于线程池和消息队列，以简化在 Android 应用中进行异步操作的代码编写。

```
private class RequestYoutubeAPI extends AsyncTask<Void, String, String>  
// AsyncTask 用于后台线程执行异步任务的类，它基于线程池和消息处理机制。
```

`AsyncTask`是一个在不需要开发者直接操作多线程和 Handler 的情况下的帮助类，适用于短时间的操作（最多几秒）。如需长时间的线程操作，建议使用多线程包`java.util.concurrent`中的功能，比如线程池。

在使用的时候，它的三个类型：`AsyncTask<Params, Progress, Result>` 描述如下：

属性	描述
Params	执行任务前，传入的参数的类型
Progress	后台线程执行的时候，用来表示进度的类型
Result	表示执行结果的类型

要使用 `AsyncTask`，必须新建一个类来继承它，并且重写 `doInBackground` 方法。通常也会重写 `onPostExecute` 方法。执行异步任务的时候，我们主要关心下面这4个方法。

方法	描述
<code>onPreExecute()</code>	执行任务前 在ui线程调用。通用用来设置任务，比如在界面上显示一个进度条。在本项目中没有使用到。
<code>Result doInBackground(Params... params)</code>	在 <code>onPreExecute()</code> 结束后立即调用这个方法。 耗时的异步任务就在这里操作 。执行任务时传入的参数会被传到这里。本项目中，用于OkHTTP的网络请求。
<code>onProgressUpdate(Progress... values)</code>	在 ui 线程中执行。后台任务还在进行的时候，这里负责处理进度信息。比如在这显示进度条动画，修改文字显示等。在本项目中没有使用到。
<code>onPostExecute(Result result)</code>	后台任务结束了调这个方法。它在 ui 线程执行 。最后的结果会传到这。本项目中，用于在这个方法中更新 UI，处理执行结果。

`doInBackground` 方法的重写（传入Void类型，返回String类型的请求结果）。

步骤如下，首先通过new OkHttpClient新建一个请求客户端，然后Build一个请求new Request.Builder().url(频道URL).build()，在请求中，最多进行MAX_ATTEMPTS次请求，每一次请求时，通过客户端创建一个newCall并执行。

`execute`是同步方法，网络请求是耗时的，因而需要放到`AsyncTask`后台线程中。

```
Response response = client.newCall(request).execute();
```

然后通过 `response.isSuccessful()` 判断是否请求成功，否则输出异常信息。

```
    @Override
    protected String doInBackground(Void... params)
    {
        OkHttpClient client = new OkHttpClient();
        // 使用okHTTP进行网络请求
        Request request = new Request.Builder().url(CHANNEL_GET_URL()).build();

        Log.d( tag: "URL2", CHANNEL_GET_URL());
        for (int retry = 0; retry < MAX_RETRIES; retry++)
        {
            try
            {
                Response response = client.newCall(request).execute();
                Log.d( tag: "URL2", msg: "请求结果" + response.toString());
                if (response.isSuccessful())
                {
                    return response.body().string();
                }
            } catch (IOException e)
            {
                Log.e( tag: "URL2", msg: "IOException: " + e.getMessage());
                e.printStackTrace();
            }
        }
        return null;
    }
```

在请求完成后，执行重写的onPostExecute(String response)方法，它接受请求的结果。这是Json字符串构成的信息，然后通过解析Json，更新数据，更新视图显示，再停止播放刷新动画：

```
swipeRefreshLayout.setRefreshing(false);
```

```
    @Override
    protected void onPostExecute(String response)
    {
        super.onPostExecute(response);
        if (response != null)
        {
            try
            {
                // doInBackground 中的response返回的数据 (String) 会传进onPostExecute(String response)
                // response转换成JSONObject即为其中的Json序列化数据
                JSONObject jsonObject = new JSONObject(response);

                // Json序列化解析
                _data = parseVideoListFromResponse(jsonObject);

                for (YoutubeDataModel item : _data)
                {
                    Log.d( tag: "URL2", item.toString());
                }
                init(_data);

            } catch (JSONException e)
            {
                e.printStackTrace();
            }
        }
        // 停止播放刷新动画
        swipeRefreshLayout.setRefreshing(false);
    }
```

其中，通过数据更新视图是在该方法中调用init(data)方法，该方法实现中，主要是重设 recyclerView 的布局管理器（线性布局），然后初始化adapter适配器对象，绑定到该Activity上，再将适配器设置到循环视图即可。适配器的实现，会放在后面介绍。

```

private void init(ArrayList<YoutubeDataModel> data)
{
    _recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
    _adapter = new VideoPostAdapter(data, getActivity());
    _recyclerView.setAdapter(_adapter);
}

```

最后，通过 `private ArrayList<YoutubeDataModel> parseVideoListFromResponse(JSONObject jsonObject)` 算法，进行Json解析。这个过程类似于剥洋葱，一层一层的往里面寻找信息。最后完成时，创建YoutubeDataModel对象，这是用来保存视频信息数据的类。

```

        JSONObject jsonSnippet = json.getJSONObject( name: "snippet");
        String title = jsonSnippet.getString( name: "title");
        String description = jsonSnippet.getString( name: "description");
        String publishedAt = jsonSnippet.getString( name: "publishedAt");
        String thumbNails = jsonSnippet.getJSONObject( name: "thumbnails").getJSONObject( name: "high").getString( name: "url");
        // 获得视频Id
        String videoId = jsonID.getString( name: "videoId");

        YoutubeDataModel youtube = new YoutubeDataModel(); 创建YoutubeDataModel
        youtube.setTitle(title);
        youtube.setDescription(description);
        youtube.setPublishedAt(publishedAt);
        youtube.setThumbNail(thumbNails);
        youtube.setVideoId(videoId);
        // 插入列表
        list.add(youtube);

```

VideoPostAdapter 视频列表适配器

该类继承自`RecyclerView.Adapter`类，泛型参数为其内部自定义类`VideoPostAdapter.YoutubePostHolder`。

定义 Adapter 时，需要替换三个关键方法：

- `onCreateViewHolder`: 每当 `Recyclerview` 需要创建新的 `viewHolder` 时，它都会调用此方法。此方法会创建并初始化 `viewHolder` 及其关联的 `view`，但不会填充视图的内容，因为 `viewHolder` 此时尚未绑定到具体数据。
- `onBindViewHolder`: `Recyclerview` 调用此方法将 `viewHolder` 与数据相关联。
- `getItemCount`: `RecyclerView` 调用此方法来获取数据集的大小

当`RecyclerView`需要展示一个新的项时，它会调用适配器的`onCreateViewHolder`方法来创建一个新的`ViewHolder`实例。一个Holder代表`RecyclerView`中的一个项目，表示一个视频。这个类`YoutubePostHolder`主要作用是绑定到`R.layout.layout_youtube_post`中的各个控件，并存储在Holder中。

```

public class VideoPostAdapter extends RecyclerView.Adapter<VideoPostAdapter.YoutubePostHolder>

```

```

@Override
public YoutubePostHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
    // LayoutInflator 是 Android 中用于将 XML 布局文件转换为实际视图的类。
    // inflate 是将 XML 布局文件转换为实际的 View 对象的过程。
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.layout_youtube_post, parent, attachToRoot: false)

    // 将view 传递给YoutubePostHolder的构造函数
    YoutubePostHolder holderView = new YoutubePostHolder((view));
    return holderView;
}

```

内部类`YoutubePostHolder`实现如下：

```
public static class YoutubePostHolder extends RecyclerView.ViewHolder
{
    TextView _textViewTitle;
    TextView _textViewDes;
    TextView _textViewData;
    ImageView _imageThumb;

    public YoutubePostHolder(@NonNull View itemView)
    {
        super(itemView);
        this._textViewTitle = (TextView)
itemView.findViewById(R.id.textViewTitle);
        this._textViewDes = (TextView)
itemView.findViewById(R.id.textViewDes);
        this._textViewData = (TextView)
itemView.findViewById(R.id.textViewData);
        this._imageThumb = (ImageView)
itemView.findViewById(R.id.imageThumb);
    }
}
```

onBindViewHolder 绑定数据与视图

在方法 `public void onBindViewHolder(@NonNull YoutubePostHolder holder, int position)` 中，会获得类内存储的YoutubeDataModel数据列表信息：

```
YoutubeDataModel dataObject = _listVideoDatas.get(position);
```

然后通过Picasso第三方库中提供的加载图片url的方法，将从数据对象中获得到的封面URL字符串信息，加载并装入到ImageView控件中：

```
Picasso.get().load(dataObject.getThumbNail()).into(imageThumb);
```

Picasso还能自动帮我们做以下事情：

- 处理Adapter 中ImageView的回收和取消下载。
- 使用最小的内存 来做复杂的图片变换。比如高斯模糊，圆角、圆形等处理。
- 自动帮我们缓存图片。内存和磁盘缓存。

使用前，需要添加依赖：

```
// Picasso 开源图片加载库
implementation 'com.squareup.picasso:picasso:2.71828'
```

设置视频项点击事件 (`holder.itemView.setOnClickListener`)，通过实例化一个匿名内部类，作为 `setOnClickListener`方法的参数，这是一种“函数式编程”的思想，实际上是为接口类型参数，填充一个实例化对象。可以类比于C++中的函数指针，C#中的委托。

点击事件需要启动视频播放页面 (`YoutubeVideoPlayActivity`，会在后面介绍)，同时向它传递信息，通过`intent.putExtra`实现。这里需要传递视频的id信息，然后启动页面。

```
holder.itemView.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View view)
    {
        // Toast.makeText(_context,"textViewTitle" ,Toast.LENGTH_LONG).show();
        String videoId = dataObject.getVideoId();

        // 启动新的 Activity, 将视频 ID 传递给该 Activity
        Intent intent = new Intent(_context, YoutubeVideoPlayActivity.class);
        intent.putExtra( name: "videoId", videoId);
        _context.startActivity(intent);
    }
});
```

视频播放 + 评论区页面

这个页面的主要代码，放在YoutubeVideoPlayActivity中。

最终效果，能够实现上方是视频小窗播放，下面是可滚动的评论区。评论区总体上也采用RecyclerView布局，每一项是一个Holder，基本与视频列表类似。

评论区理论上能够显示评论发布者的id，头像，发布时间，评论内容等。但是这里从简化处理，省略了头像的显示，如果希望实现，其实与视频封面的显示同理，可以通过Picasso进行get。

中午12:02

VPN HD HD 99%

猪猪视频



@2enjoyk253

加油👏.....

2023-12-11T05:39:41Z

@MingEvan

計劃總是落後於市場力量的有一實証

2023-11-11T06:47:40Z

@Sharonli23345

足协正在光速赶过来

2023-09-03T21:01:19Z

@bozhang938

这个节目比其他两个在说“足协打”来得 大胆 生宣 亚4K

这个找了几下却发现没有类似。但观比赛，晋级赛，循环赛阶段，关注的人不多，或许就在校园球场就已经完成比赛了。到决赛阶段，胜利的一支球队，奖品或许只是几箱啤酒而已。这才是中国文化复兴该有的样子。不是勾栏那般的高雅，也不需要太多“高水平高标准”的“专家”来指导。村超足球突然网络爆发，或许是国办足协及商办足球俱乐部荼毒久矣，普通球迷甚至路人已经极度失望之余，野性回归的一种选择，或许这波新鲜劲过去，类似今年“村超”的盛况会消退，但人家原本就只是村民业余的体育活动，如人饮水，即使没有

在实现的时候，大部分代码和视频列表部分类似。需要实现 `videoCommentAdapter`，用作评论适配器；实现内部类 `CommentHolder`，用于每条评论视图控件的绑定；`CommentDataModel` 类，用作评论数据的存储；在 `YoutubeVideoPlayActivity` 中，也需要实现 `VIDEO_COMMENTS_URL()` 获得评论Url，`RequestComment` 类用于异步网络请求，`parseCommentListFromJson` 用于解析评论信息等。

AsyncTask替代品：封装Thread

实际上`AsyncTask`已经被废弃。因此在评论获取的过程中，希望使用`Thread`多线程替代`AsyncTask`。后者的主要问题包括由于`AsyncTask`与`Activity`或`Fragment`的生命周期无关而导致的内存泄漏等等。

因而我简单封装了`Thread`执行子线程任务的函数为一个抽象类`BackgroundTask`，其命名规范与`AsyncTask`完全一致。

```
public abstract class BackgroundTask
{
    /**
     * 与此后台任务关联的 Activity。
     */
    private Activity _activity;

    /**
     * 使用指定的 Activity 构造一个新的 BackgroundTask 实例。
     */
    public BackgroundTask(Activity activity) { this._activity = activity; }

    /**
     * 在后台线程中调用 doInBackground 方法，然后在主线程中调用 onPostExecute。
     */
    private void startBackground()
    {
    }

    /**
     * 执行后台任务的入口方法，由子类实现。
     */
    public abstract void doInBackground();

    /**
     * 后台任务执行完成后在主线程中调用的方法，由子类实现。
     */
    public abstract void onPostExecute();
}

/**
 * 执行后台任务入口，启动后台线程执行 doInBackground 方法。
 */
public void execute() { startBackground(); }
```

在启动子线程任务的时候，首先启动后台子线程。然后使用隶属于Activity的方法runOnUiThread()，这个方法是运行于主线程中，可以进行UI的更新。同样也是传递一个Runnable的匿名内部类并实现run()方法。

```
/** 在后台线程中调用 doInBackground 方法，然后在主线程中调用 onPostExecute。 */
1个用法
private void startBackground()
{
    // 实现一个匿名内部类，传递给接口参数，“函数式编程”
    new Thread(new Runnable()
    {
        public void run()
        {
            // 首先在子线程启动后台任务
            doInBackground();
            // 然后在主线程中调用 onPostExecute 方法
            _activity.runOnUiThread(new Runnable()
            {
                public void run() { onPostExecute(); }
            });
        }
    }).start();
}
```

评论数据请求的类定义为：

```
private class RequestComment extends BackgroundTask
```

视频播放页面布局：NestedScrollView 嵌套 RecyclerView

顶层布局使用NestedScrollView，然后内部是一个RelativeLayout，再往内部嵌套了用于视频播放的WebView 和 评论区显示的RecyclerView。

NestedScrollView其作用就是作为控件父布局，从而具备（嵌套）滑动功能。ScrollView嵌套RecyclerView存在滑动冲突，显示不全的问题。NestedScrollView嵌套RecyclerView不会存在显示问题。

视频播放WebView 与 iFrame

播放视频采用WebView嵌入HTML语言的方式实现。这是因为需要调用Youtube提供的iFrame框架，实现获取Youtube视频源。其中HTML示例代码如下：

```
<!DOCTYPE html>
<html>
<body>
<div id="player"></div>
<script>
var tag = document.createElement('script');

tag.src = "https://www.youtube.com/iframe_api";
var firstScriptTag = document.getElementsByTagName('script')[0];
firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

var player;
function onYouTubeIframeAPIReady() {
    player = new YT.Player('player', {
        height: '200',
```

```

        width: '400',
        videoId: 'qZ4UEhFEzvI',
        events: {
          'onReady': onPlayerReady,
          'onStateChange': onPlayerStateChange
        }
      });

    }

    function onPlayerReady(event) {
      event.target.playVideo();
    }

    var done = false;
    function onPlayerStateChange(event) {
      if (event.data == YT.PlayerState.PLAYING && !done) {
        setTimeout(stopVideo, 6000);
        done = true;
      }
    }
    function stopVideo() {
      player.stopVideo();
    }
  
```

1. `onYouTubeIframeAPIReady` 函数会在播放器 API 代码下载后马上运行，构造视频播放器对象。
2. `onPlayerReady` 在 `onReady` 事件触发时运行
3. API会在播放器状态改变时调用 `onPlayerStateChange` 函数，指示播放器正在播放、已暂停、已完成等等。

使用iFrame的好处，可以方便的解析视频源，获取Youtube提供的视频播放器功能（倍速，暂停，快进，画质设置等）且完美兼容嵌入到APP中。

横屏全屏播放

上述过程中，使得WebView和评论区RecycleView在一个滚动布局下。但是问题是如果希望横屏时，能够实现类似于全屏播放的效果，可能需要再开一个Activity。这样做会带来很多麻烦，因此为了避免这个问题，采用重设布局参数的方式解决。

当传感器检测到横屏时，能够自动将WebView的布局重设，并重新加载Landscape摸索下的HTML（长宽参数改变），同时“隐藏”评论区视，隐藏状态栏和APP信息栏。

在视频播放界面创建的时候，需要做如下设置：

```

// 获取状态栏
// getSupportActionBar 是 父类AppCompatActivity的函数
actionBar = getSupportActionBar();

// 获取传递过来的视频 ID
// getIntent()是最顶层父类Activity中的函数
_videoId = getIntent().getStringExtra("videoId");

// 设置 WebView
_webView.getSettings().setJavaScriptEnabled(true);

```

```
// 加载url, data_portait是默认竖屏下的HTML脚本  
_webView.loadDataWithBaseURL(null, data_portait, "text/html", "UTF-8", null);
```

同时需要设置WebView全局布局监听事件，创建匿名内部类，并重写onGlobalLayout方法。

```
_webView.getViewTreeObserver().addOnGlobalLayoutListener(new ViewTreeObserver.OnGlobalLayoutListener() {
```

通过getResources() 在检测到横屏时，设置全屏标记，隐藏状态栏，移除评论区ViewGroup，并重新加载横屏下的WebViewURL，这里使用data_landscape。

```
if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE) {  
    if (actionBar != null) {  
        actionBar.hide(); // 隐藏状态栏  
    }  
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN); // 设置全屏标志  
  
    // 移除评论区  
    if (recyclerView.getParent() != null) {  
        ((ViewGroup) recyclerView.getParent()).removeView(recyclerView);  
    }  
  
    _webView.loadDataWithBaseUrl(baseUrl: null, data_landscape, mimeType: "text/html", encoding: "UTF-8", historyUrl: null);  
}  
else {
```

```
} else {  
    // 坚屏  
    if (actionBar != null) {  
        actionBar.show(); // 显示状态栏  
    }  
    getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN); // 隐藏全屏标志  
  
    if (recyclerView.getParent() != null) {  
        ((ViewGroup) recyclerView.getParent()).removeView(recyclerView);  
    }  
  
    layout.addView(recyclerView, recyclerViewLayoutParams);  
  
    _webView.setLayoutParams(new RelativeLayout.LayoutParams(  
        RelativeLayout.LayoutParams.MATCH_PARENT, RelativeLayout.LayoutParams.WRAP_CONTENT));  
}
```

最终效果如下，在横屏后自动检测并将视频播放状态变为全屏模式。

同时，下图展示了Iframe提供的Youtube播放服务，包括画质，播放速度等。



尾声

在仿Youtube视频播放实现过程中，比较创新性，有挑战性的实现了Googleapis爬取视频功能，并加入刷新功能。在刷新功能实现后，甚至可以做随机推荐和算法推荐，即推送用户可能喜欢的视频。同时这里也可以加入通过标签和标题进行视频搜索播放功能，后续可以进行探究。

然后，添加了iFrame框架，嵌入WebView HTML 语言的方式，实现了比较成熟的视频播放器。具体参数可以在HTML中设置。并且，加入NestedScrollView，解决嵌套滚动冲突问题，很好的实现了滚动评论区。

最后，比较创新的实现了全屏播放效果和竖屏之间切换的逻辑，注重了用户体验性。

合作开发

开发过程中，使用了sourceTree图形化版本管理软件，和`ydy`。

