计算机科学与技术学院课程设计成绩单

课程名称：移动平台软件设计　　指导教师：张智

姓名	卢俊朴	性别	男	学号	202113407120	班级	软件工程 2101
综合成绩		成绩等级					
程序运行情况（30 分）							
对问题的答辩情况（30 分）							
学生工作态度与独立工作能力（20 分）							
设计报告规范性（20 分）							

百分制成绩与等级制成绩对应表

百分制	90-100	85-89	82-84	78-81	75-77	72-74	68-71	64-67	60-63	
等级制	A	A-	B+	B	B-	C+	C	C-	D	F

计算机科学与技术学院

课 程 设 计 报 告

课程名称：移动平台软件设计

专　　业：软件工程

班　　级：21 级 2101 班

学　　号：202113407120

姓　　名：卢俊朴

指导老师：张智

目录

（1）系统概述

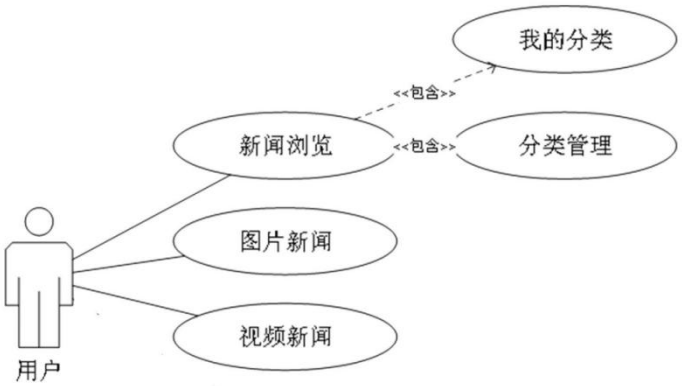
近年来，随着科学技术的飞速发展，信息的传播方式和手段也逐渐现代化。依托互联网等新媒体已经变得更加高效便捷。新的传播方式和媒介逐渐改变了传统的信息传播方式。同时，对新媒体环境下新闻载体的研究也被赋予了鲜明的时代特征。例如目前大部分网民都是使用移动设备来接入互联网的，因此新闻这一媒体产品的载体已经从过去的传统媒体转到了移动端媒体。

本系统选择使用 Java 语言来设计一个基于 Android 的新闻客户端。用户通过此 APP 可以实现，访问查看实时的新闻资讯、图片和视频，并可以详细查看内容。

（2）系统设计

用户用例

本文系统包含的功能有：新闻图文浏览、分类浏览、分类管理、图片新闻和视频新闻等，通过用户用例图的形式展示如下：



图一 安卓新闻客户端用例图

功能模块及各个模块详细功能

在系统的开发之前，需要对需求分析的产出进行一个设计，这就是系统的总体设计，总体设计需要考虑系统的界面、功能等方面的内容，根据系统的功能要求，本文将系统的功能以功能模块图的形式来展示，如图 2 所示：

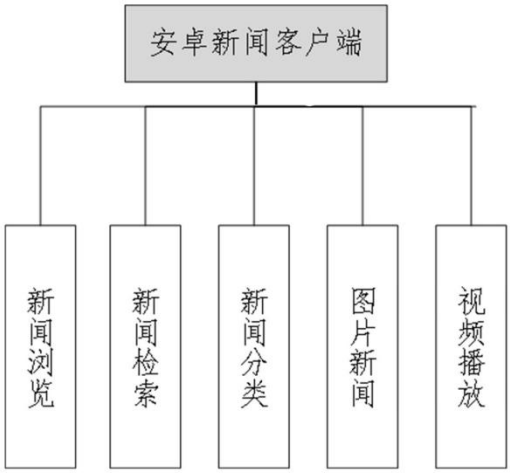


图2 系统功能模块图

新闻游览模块：显示实时的对应新闻频道中的各种新闻(网易新闻中的新闻)的封面，标题，发布日期，发布者等信息。

新闻检索模块：点击单个新闻，可以显示具体新闻内容(以 html 形式展现，包括标题，发布日期，发布者，具体内容和图片)

新闻分类模块：对各种类型新闻新闻进行分类，包括头条、要闻、科技、财经，明星等等非常多频道，用户可在新闻页面切换频道，也可以进行频道管理，添加推荐分类中的频道在新闻页面。

图片新闻模块：显示一系列图片封面，点击可滑动查看所有图片。

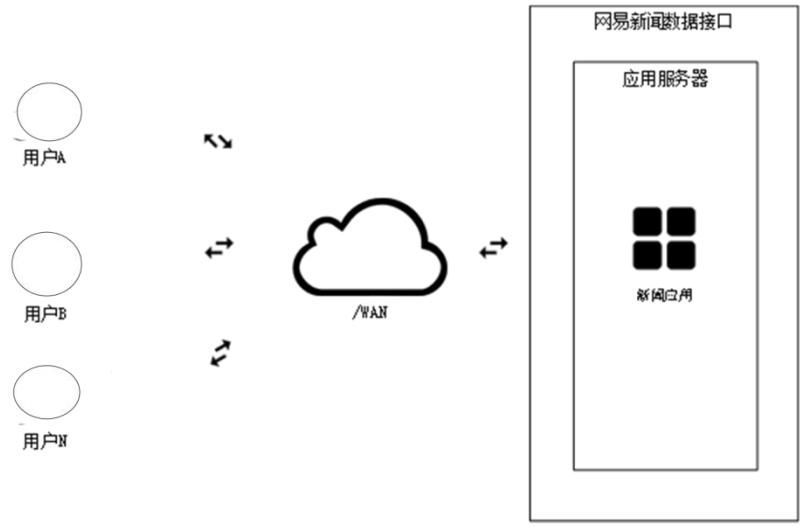
视频播放模块：随机显示近期热门视频，显示视频封面，标题，发布日期，发布

者等信息，点击后播放视频。

（3）系统实现（使用流程图、算法说明托来描述）

注：解释新闻模块的实现，这是最为复杂的模块，其他模块的实现大多与此模块相似

一、数据获取模块



URL(Api.java)

介绍：

抓取网易新闻的栏目，新闻，图片，和视频的 url。

Url 由抓取的前缀（区分频道），后缀/结尾（从哪里开始获取），详细链接（tid……）组成。

通过抓取网易新闻 api，再包装成 Url，封装成 Api。包括了基础的 API 地址、频道 ID、起始索引和其他参数用于后续的网络请求，以获取新闻数据。

主要代码：

```
/**
 * 抓取网易新闻 api
 * URL 示例：
 * 栏目列表：
 * http://c.m.163.com/nc/topicset/android/subscribe/manage/listspecial.html

 *  其他类型的图片 api：
 *  http://c.m.163.com/photo/api/list/0096/54GI0096.json
**/
public class Api {
//前缀
    public static final String host = "http://c.m.163.com/";
    // 普通栏目前缀，示例：
http://c.m.163.com/nc/article/list/T1467284926140/0-20.html
    public static final String CommonUrl = host + "nc/article/list/";
    // 图片栏目的前缀， 示例：
http://pic.news.163.com/photocenter/api/list/0031/6LRK0031,6LRI0031/0/20.json
    // 推荐图片：0031， 新闻图片： 0001 ， 明星图片：0003
    public static final String PictureUrl =
"http://pic.news.163.com/photocenter/api/list/";
    // 特殊频道前缀：热点、网易号适用
    public static final String SpecialColumn1 = "recommend/getSubDocPic?";
    // 特殊频道前缀: 视频/段子、美女、萌宠适用
    public static final String SpecialColumn2 =
"recommend/getChanListNews?channel=";
    // 文章详情前缀
    public static final String DetailUrl = host + "nc/article/";
//后缀
    // 普通新闻栏目的结尾，注意要在前面添加新闻从那一条开始获取
    // 如 ：http://c.m.163.com/nc/article/list/T1467284926140/0-20.html，获取最新
的 20 条新闻
    // http://c.m.163.com/nc/article/list/T1467284926140/20-20.html，获取从第 20 条
开始的后面 20 条新闻
    public static final String endUrl = "-20.html";
```

```
    // 所有特殊频道的结尾，如：热点、网易号、段子、美女、萌宠
    public static final String SpecialendUrl = "&size=10&offset=";
    /**
     * 新闻文章详情页链接尾部
     * 示例：http://c.m.163.com/nc/article/C8T2G5QL0511A99L/full.html
     * 其中 postId ： C8T2G5QL0511A99L
     */
   public static final String endDetailUrl = "/full.html";

    /**
     * 图片新闻详情页连接
     * 链接示例： http://c.m.163.com/photo/api/set/0031/13897.json
     * 图片详情页：0031 为图片频道（列表连接后面的 0031），13897 为图片新闻 setid
     */
    public static final String PictureDetailUrl = host + "photo/api/set/";
    // 图片新闻详情页结尾
    public static final String endPictureDetailUrl = ".json";
    /**
     * 视频详情链接
     *  示例： http://c.m.163.com/nc/video/detail/VC5BA022H.html
     *  其中 VC5BA022H 是视频的 vid
     */
    public static final String videoDetailUrl = host + "nc/video/detail/";
    // 视频详情链接尾部
    public static final String EndUrlVideoDetailUrl = ".html";
   ……
    // NBA
    public static final String NBAId = "T1348649145984";
   ……

    // 图片新闻尾部，需要在签名添加参数，可获得从某条新闻之后的 20 条新闻
    // 示例 ：
http://pic.news.163.com/photocenter/api/list/0001/00AN0001,00AO0001/0/20.json
    public static final String endPicture = "/20.json";
```

```
    // 图片
    public static final String specialPictureId = "T1419316384474";

    // 推荐图片：0031/6LRK0031,6LRI0031/      应使用瀑布流
    public static final String RecommendPictureId = "0031";
    public static final String RecommendPicture = "/6LRK0031,6LRI0031/";

    // 新闻图片：0001/00AP0001,3R710001,4T8E0001/      横向排版
    public static final String NewsPictureId = "0001";
    public static final String NewsPicture = "/00AP0001,3R710001,4T8E0001/";
    // 热点图片：0001/00AN0001,00AO0001/       横向排版
    public static final String HotPicture = "/00AN0001,00AO0001/";

    // 明星图片：
0003/00AJ0003,0AJQ0003,3LF60003,00B70003,00B50003/         瀑布流排版
    public static final String StarPictureId = "0003";
    public static final String StarPicture =
"/00AJ0003,0AJQ0003,3LF60003,00B70003,00B50003/";

    /**
     *    特殊格式新闻：
     *    下面几类连接基本一致：
     *    热点、网易号类的新闻连接形式差不多，但是网易号多了 from 参数
     *    使用 getSubDocPic
     *    段子、美女、萌宠连接基本一致
     *    使用  getChanListNews
     **/
    // 热点
    public static final String specialRedianId = "T1427878984398";
    // 热点 url
    public static final String RedianUrl = host + SpecialColumn1 + SpecialendUrl;
    // 网易号
    public static final String specialSubId = "T1449126525962";
    // 网易号 url
```

```
    public static final String WangYiHaoUrl = host + SpecialColumn1 +
"from=netease_h" +SpecialendUrl;
    // 下列为特殊频道 URL，  统一连接形式为：
    //
http://c.m.163.com/recommend/getChanListNews?channel=T1419316284722&size=
10&offset=
    public static final String specialURL = host + SpecialColumn2;
    // 视频
    //http://c.m.163.com/recommend/getChanListNews?channel=T1457068979049&o
ffset=0&size=20&devId=44t6%2B5mG3ACAOlQOCLuIHg%3D%3D
    public static final String specialVideoId = "T1457068979049";
    // 手机开发 id 号，任意手机的开发 Id 号都可以
    public static final String devId =
"&devId=44t6%2B5mG3ACAOlQOCLuIHg%3D%3D";
    // 段子
    public static final String specialJokeId = "T1419316284722";
    // 美女
    public static final String specialGirlId = "T1456112189138";
    // 热点视频  http://c.m.163.com/nc/video/list/V9LG4B3A0/n/10-10.html
    public static final String Video = host + "nc/video/list/";
    public static final String VideoCenter = "/n/";
    public static final String videoEndUrl = "-20.html";
    // 热点视频
    public static final String VideoReDianId = "V9LG4B3A0";
    // 推荐视频
//http://c.m.163.com/recommend/getChanListNews?channel=T1457068979049&offse
t=0&size=20&devId=44t6%//2B5mG3ACAOlQOCLuIHg%3D%3D
    public static final String VideorecommendId = "T1457068979049";
   ……
    public static final String VideoLive =
"http://data.live.126.net/livechannel/previewlist.json";
}
```

执行 HTTP/HTTPS 请求

1. 对是否支持 HTTPS 进行相关配置

```
public class HttpClientFactory {
    /** http 请求最大并发连接数 ，即同时可以执行的最大 HTTP 请求数量*/
    private static final int MAX_CONNECTIONS = 10;
    /** 超时时间。表示如果一个 HTTP 请求在 10 秒内没有完成，它将会超时 */
    private static final int TIMEOUT = 10 * 1000;
    /** 缓存大小 ，8 * 1024 字节，即 8KB*/
    private static final int SOCKET_BUFFER_SIZE = 8 * 1024;

    public static DefaultHttpClient create(boolean isHttps) {//是否支持 HTTPS 协议
//        HTTPParams 对象包含了一系列的 HTTP 连接参数的配置，如连接超时时间、Socket 超时时间、缓存大小等
        HttpParams params = createHttpParams();
        DefaultHttpClient httpClient = null;
        if (isHttps) {//支持 http 与 https 进行 HTTPS 的相关配置
            //  注册支持的协议
            SchemeRegistry schemeRegistry = new SchemeRegistry();
            schemeRegistry.register(new Scheme("http",
PlainSocketFactory.getSocketFactory(), 80));
            schemeRegistry.register(new Scheme("https",
SSLSocketFactory.getSocketFactory(), 443));
            // ThreadSafeClientConnManager 线程安全管理类 用于线程安全地管理连接
            ThreadSafeClientConnManager cm = new
ThreadSafeClientConnManager(params, schemeRegistry);
            httpClient = new DefaultHttpClient(cm, params);
        } else {
            //如果 isHttps 为 false，表示只支持 HTTP 协议，不进行 HTTPS 相关的配置。
            httpClient = new DefaultHttpClient(params);
        }
        return httpClient;
```

```
    }

    private static HttpParams createHttpParams() {
        final HttpParams params = new BasicHttpParams();
        // 设置是否启用旧连接检查，默认是开启的。关闭这个旧连接检查可以提高一点点性能，但是增加了 I/O 错误的风险（当服务端关闭连接时）。
        // 开启这个选项则在每次使用老的连接之前都会检查连接是否可用，这个耗时大概在 15-30ms 之间
        HttpConnectionParams.setStaleCheckingEnabled(params, false);
        HttpConnectionParams.setConnectionTimeout(params, TIMEOUT);// 设置链接超时时间
        HttpConnectionParams.setSoTimeout(params, TIMEOUT);// 设置 socket 超时时间
        HttpConnectionParams.setSocketBufferSize(params, SOCKET_BUFFER_SIZE);// 设置缓存大小
        HttpConnectionParams.setTcpNoDelay(params, true);// 是否不使用延迟发送(true 为不延迟)
        HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1); // 设置协议版本
        HttpProtocolParams.setUseExpectContinue(params, true);// 设置异常处理机制
        HttpProtocolParams.setContentCharset(params, HTTP.UTF_8);// 设置编码
        HttpClientParams.setRedirecting(params, false);// 设置是否采用重定向

        ConnManagerParams.setTimeout(params, TIMEOUT);// 设置超时
        ConnManagerParams.setMaxConnectionsPerRoute(params, new ConnPerRouteBean(MAX_CONNECTIONS));// 多线程最大连接数
        ConnManagerParams.setMaxTotalConnections(params, 10); // 多线程总连接数
        return params;
    }

}
```

2. HttpClient 链接封装

介绍：

通过如下 3 步来访问 HTTP 资源。

1. 创建 HttpGet 或 HttpPost 对象，将要请求的 URL 通过构造方法传入 HttpGet 或 HttpPost 对象。

2. 使用 DefaultHttpClient 类的 execute 方法发送 HTTP GET 或 HTTP POST 请求，并返回 HttpResponse 对象。

3. 通过 HttpResponse 接口的 getEntity 方法返回响应信息，并进行相应的处理。

如果使用 HttpPost 方法提交 HTTP POST 请求，则需要使用 HttpPost 类的 setEntity 方法设置请求参数。参数则必须用 NameValuePair[]数组存储。

具体代码

主要代码以及详细注解

```
//表示不检测过期的方法,不显示使用了不赞成使用的类或方法时的警告
@SuppressWarnings("deprecated")
public class HttpHelper {

    private static final String TAG = HttpHelper.class.getSimpleName();

    /**
     * get 请求，获取返回的字符串内容
     */
    public static void get(String url, HttpCallbackListener httpCallbackListener) {
        HttpGet httpGet = new HttpGet(url);
        //调用  execute()  方法执行 HTTP 请求
        execute(url, httpGet, httpCallbackListener);
    }

    /**
     * post 请求，向服务器提交数据。
     */
    public static void post(String url, byte[] bytes, HttpCallbackListener
httpCallbackListener) {
        HttpPost httpPost = new HttpPost(url);
```

```
        ByteArrayEntity byteArrayEntity = new ByteArrayEntity(bytes);
        httpPost.setEntity(byteArrayEntity);
        execute(url, httpPost, httpCallbackListener);
    }

    /**
     * 下载
     */
    public static void download(String url, HttpCallbackListener httpCallbackListener)
{
        HttpGet httpGet = new HttpGet(url);
        execute(url, httpGet, httpCallbackListener);
    }

    /**
     * 执行网络访问
     */
    private static void execute(String url, HttpRequestBase requestBase,
HttpCallbackListener httpCallbackListener) {
        //检查 URL 是否以 "https://" 开头来判断是否需要采用 HTTPS 协议。
如果 URL 以 "https://" 开头
        boolean isHttps = url.startsWith("https://");
        //则创建一个支持 HTTP 和 HTTPS 协议的客户端(HttpClientFactory 中)
        AbstractHttpClient httpClient = HttpClientFactory.create(isHttps);
        //创建一个 HttpContext 对象，用于管理 HTTP 上下文信息
        HttpContext httpContext = new SyncBasicHttpContext(new
BasicHttpContext());
        //获取 HTTP 客户端的请求重试处理器。这是用于处理网络请求失败后的
重试机制
        HttpRequestRetryHandler retryHandler =
httpClient.getHttpRequestRetryHandler();
        int retryCount = 0;//初始化重试次数计数器，用于记录重试的次数
        boolean retry = true;//初始化重试标志，表示是否需要继续重试
        while (retry) {
            try {
```

```
            //使用 httpClient 执行 requestBase 所代表的 HTTP 请求(GET。POST……)，访问网络，并将响应存储在 response 变量中。
                HttpResponse response = httpClient.execute(requestBase, httpContext);
                //获取 HTTP 响应的状态码，以便后续处理
                int stateCode   = response.getStatusLine().getStatusCode();
//                  LogUtils.e(TAG, "http 状态码：" + stateCode);
                if (response != null) {
                    //如果 HTTP 响应状态码为 200（HTTP_OK），表示请求成功，进入处理成功的分支
                    if (stateCode == HttpURLConnection.HTTP_OK){
                        //进一步处理响应数据   获取返回的字符串或者流
                        HttpResult httpResult = new HttpResult(response, httpClient, requestBase);
                        String result = httpResult.getString();
                        //响应数据为空，则抛出运行时异常，表示数据为空
                        if (!TextUtils.isEmpty(result)){
                            httpCallbackListener.onSuccess(result);
                            return;
                        } else {
                            throw new RuntimeException("数据为空");
                        }
                    } else {//如果响应状态码不是 200（HTTP_OK），则抛出运行时异常，表示请求失败
                        throw new RuntimeException(HttpRequestCode.ReturnCode(stateCode));
                    }
                }
            } catch (Exception e) {//捕获异常，表示在执行 HTTP 请求时发生了错误。这可能包括网络连接问题、超时等
                IOException ioException = new IOException(e.getMessage());
                //将异常交给重试机制，并检查是否需要继续重试。如果需要重试
                retry = retryHandler.retryRequest(ioException, ++retryCount, httpContext);
                LogUtils.e(TAG, "重复次数：" + retryCount + "    :"+ e);
                if (!retry){
```

```
                    httpCallbackListener.onError(e);
                }
            }
        }
    }

    /**
     * http 的返回结果的封装，可以直接从中获取返回的字符串或者流
     */
    public static class HttpResult {
        private HttpResponse mResponse;//用于存储 HTTP 响应对象，包括响应头和响应体
        private InputStream mIn;//存储响应体的输入流。这个输入流用于从响应体中读取原始数据。
        private String mStr;//成员变量 mStr 用于存储响应数据的字符串形式。这个字符串是从输入流中读取的，并且在第一次获取后会被缓存
        private HttpClient mHttpClient;//用于存储 HTTP 客户端对象。这个对象在构造 HttpResult 时传递进来，用于处理 HTTP 请求
        private HttpRequestBase mRequestBase;//用于存储 HTTP 请求对象
        public HttpResult(HttpResponse response, HttpClient httpClient, HttpRequestBase requestBase) {
            mResponse = response;
            mHttpClient = httpClient;
            mRequestBase = requestBase;
        }
        //用于获取 HTTP 响应的状态码
        public int getCode() {
            StatusLine status = mResponse.getStatusLine();
            return status.getStatusCode();
        }


        /**
         * 从结果中获取字符串，一旦获取，会自动关流，并且把字符串保存，方便下次获取
```

```
     */
    public String getString() {
        //方法会检查成员变量 mStr 是否已经存储了响应数据的字符串。如果已经存储了，说明之前已经获取过响应数据，
        // 就直接返回缓存的字符串，避免重复获取
        if (!StringUtils.isEmpty(mStr)) {
            return mStr;
        }
        //如果 mStr 为空（即没有缓存数据），则调用 getInputStream() 方法获取响应的输入流。getInputStream()
        // 方法用于获取响应体的输入流，这是响应数据的原始形式
        InputStream inputStream = getInputStream();
        //创建一个 ByteArrayOutputStream 对象 out，它用于将输入流中的数据写入内存中的字节数组
        ByteArrayOutputStream out = null;
        if (inputStream != null) {
            try {
                //接收从输入流中读取的数据
                out = new ByteArrayOutputStream();
                //创建一个字节数组 buffer，用于临时存储从输入流读取的数据。
                byte[] buffer = new byte[1024 * 4];
                int len = -1;//用于记录每次从输入流中读取的字节数
                //从输入流中读取数据并写入 out 流。循环会一直执行，直到读取完所有数据。
                while ((len = inputStream.read(buffer)) != -1) {
                    //成功读取数据，就将 buffer 中的数据写入 out 流，从而将数据存储在内存中
                    out.write(buffer, 0, len);
                }
                //将 out 流中的数据转换为字节数组 data。
                byte[] data = out.toByteArray();
                mStr = new String(data, "utf-8");
            } catch (Exception e) {
                LogUtils.e(TAG, e);
            } finally {
```

```
                //关闭 out 流，释放资源。
                IOUtils.close(out);
                close();
            }
        }
        return mStr;
    }


    /**
     * 获取流，需要使用完毕后调用 close 方法关闭网络连接
     */
    public InputStream getInputStream() {
        if (mIn == null && getCode() < 300) {
            HttpEntity entity = mResponse.getEntity();
            try {
                mIn = entity.getContent();
            } catch (Exception e) {
                LogUtils.e(TAG, e);
            }
        }
        return mIn;
    }

    /**
     * 关闭网络连接
     */
    public void close() {
        if (mRequestBase != null) {
            mRequestBase.abort();
        }
        IOUtils.close(mIn);
        if (mHttpClient != null) {
      mHttpClient.getConnectionManager().closeExpiredConnections();
        }
```

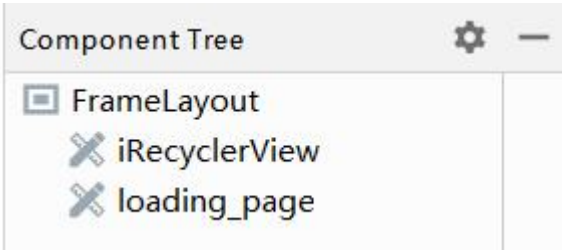
```
        }
    }
}
```

新闻数据处理

介绍：

获取传入的频道 ID（tid）和设置请求 URL。
处理数据的加载、刷新和展示，通过网络请求获取新闻数据。
用于展示新闻列表，支持下拉刷新和上拉加载更多功能，以及处理缓存数据的展示和网络数据的请求。

前端：



IRecyclerView 用于显示大量数据的 Android UI 组件，特别适用于需要滚动的列表和网格布局
LoadMoreFooterView 是一个自定义的下拉刷新头部视图，用于指示滚动时加载更多内容的视图。

后端主要代码以及详细解释

```
public class NewsListFragment extends BaseFragment {


    //表示 NewsListFragment 类的简单名称的常量字符串。通常用于日志记录和调试。
    private final String TAG = NewsListFragment.class.getSimpleName();
    private static final String KEY = "TID";     //线程值，用作在片段或活动之间传递数据的键
    private String mUrl;          // 请求网络的 url
    private String tid;  // 表示频道或类别的 ID。
```

```
    private View mView;       // 布局视图
//     用于处理新闻列表的关键 UI 组件，它与适配器一起负责展示新闻数据并允许用户滚动和查看更多内容
    private IRecyclerView mIRecyclerView; //用于显示大量数据的 Android UI 组件，特别适用于需要滚动的列表和网格布局
    private LoadMoreFooterView mLoadMoreFooterView;//用于指示滚动时加载更多内容的视图。
    private NewsListAdapter mNewsListAdapter;//用于将数据填充到 mIRecyclerView 中的适配器
    private LoadingPage mLoadingPage;//负责显示加载、空白和错误状态的视图或组件。

    private List<NewsListNormalBean> mNewsListNormalBeanList;    // 启动时获得的数据
    private List<NewsListNormalBean> newlist;    // 上拉刷新后获得的数据

    private int mStartIndex = 0;     // 请求数据的起始参数
    public ThreadManager.ThreadPool mThreadPool; // 线程池
    private boolean isPullRefresh;   // 判断当前是下拉刷新还是上拉刷新
    private boolean isShowCache = false; // 是否有缓存数据被展示

    private boolean isConnectState = false;   // 判断当前是否在联网刷新, false 表示当前没有联网刷新

    //Handler 负责根据接收到的消息类型来更新 UI。
    // 处理不同消息，包括展示新闻、错误信息、下拉刷新加载更多等操作。
    private Handler mHandler = new Handler(new Handler.Callback() {
        @Override
        public boolean handleMessage(Message message) {
            int what = message.what;
            String result;
            String error;
            switch (what) {
                case HANDLER_SHOW_NEWS:
                    bindData();           //填充 UI 中的新闻数据
```

```
                    showNewsPage();       //显示新闻页面
                    break;
                case HANDLER_SHOW_ERROR:
                    error = (String) message.obj;
                    ToastUtils.showShort(error);
                    // 如果有缓存内容就不展示错误页面
                    if (!isShowCache) {
                        showErroPage();
                    }
                    break;
            ……              }
            return false;
        }
    });

//从外部往 Fragment 中传参数的方法    将数据传递给新的 Fragment 实例
//用这种方式来创建 Fragment 并向其传递参数，以便在 Fragment 内部使用这些参数进行特定的操作和数据加载。
    public static NewsListFragment newInstance(String tid) {
        Bundle bundle = new Bundle();
        bundle.putSerializable(KEY, tid);
        NewsListFragment fragment = new NewsListFragment();
        fragment.setArguments(bundle);
        return fragment;
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        //初始化布局，初始化视图、加载数据、设置监听器
        mView = inflater.inflate(R.layout.fragment_news_list, container, false);
        initView();
        initValidata();
        initListener();
```

```
        LogUtils.d(TAG, "调用了 onCreateView" + tid);
        return mView;
    }
    @Override
    public void initView() {
        //获取布局元素引用
        mLoadingPage = (LoadingPage) mView.findViewById(R.id.loading_page);
        mIRecyclerView = (IRecyclerView)
mView.findViewById(R.id.iRecyclerView);

        //设置 RecyclerView 的布局管理器为 LinearLayoutManager，这表示列表项将按线性方式排列
        mIRecyclerView.setLayoutManager(new
LinearLayoutManager(getActivity()));
        //添加一个分割线，在列表项之间添加分隔线
        mIRecyclerView.addItemDecoration(new
DividerGridItemDecoration(getActivity()));

        //在用户下拉列表时显示
        mLoadMoreFooterView = (LoadMoreFooterView)
mIRecyclerView.getLoadMoreFooterView();
        ClassicRefreshHeaderView classicRefreshHeaderView = new
ClassicRefreshHeaderView(getActivity());
        classicRefreshHeaderView.setLayoutParams(new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
DensityUtils.dip2px(getActivity(), 80)));
        //设置了这个自定义的下拉刷新头部视图到 mIRecyclerView 中，以便下拉列表时，可以被正确地显示
        mIRecyclerView.setRefreshHeaderView(classicRefreshHeaderView);
        //展示加载页面的内容
        showLoadingPage();
    }

    @Override
    public void initValidata() {
```

```
        if (getArguments() != null) {//取得是 bundle 里封装的 tid
            //取出保存的频道 TID
            tid = getArguments().getString("TID");
        }
        // 创建线程池 用于管理和执行后续的网络请求和数据处理任务
        mThreadPool = ThreadManager.getThreadPool();
        //包括了基础的 API 地址、频道 ID、起始索引和其他参数,用于后续的网络请求，以获取新闻数据
        mUrl = Api.CommonUrl + tid + "/" + mStartIndex + Api.endUrl;
        //从本地缓存中获取新闻数据 ，如果已经有缓存的数据可用，可以在显示新闻列表之前尝试展示缓存的数据
        getNewsFromCache();
    }

    /**
     * 从缓存中读取并解析显示数据
     */
    private void getNewsFromCache() {
        //它在一个新的线程中执行操作，以避免在主线程中进行耗时的文件读取操作。
        mThreadPool.execute(new Runnable() {
            @Override
            public void run() {
                //尝试从本地缓存中读取新闻数据
                String cache = LocalCacheUtils.getLocalCache(mUrl);
                if (!TextUtils.isEmpty(cache)) {
                    //成功从缓存中读取到数据 将其转换为新闻列表对象
                    mNewsListNormalBeanList = DataParse.NewsList(cache, tid);
                    if (mNewsListNormalBeanList != null) {
                        LogUtils.d(TAG, "读取缓存成功");
                        //表示已展示缓存数据
                        isShowCache = true;
                        //通知展示缓存数据
                        Message message = mHandler.obtainMessage();
                        message.what = HANDLER_SHOW_NEWS;
```

```
                    //使用 Handler 发送一条消息（HANDLER_SHOW_NEWS）给主线程，通知主线程展示新闻数据
                    mHandler.sendMessage(message);
                } else {
                    isShowCache = false;
                }
            }
            //检查是否需要联网刷新
            if (!isLastNews(tid) || TextUtils.isEmpty(cache)) {
                // 先判断当前缓存时间是否超过 3 个小时，超过则联网刷新
                if (NetWorkUtil.isNetworkConnected(getActivity())) {
                    // 有网络的情况下请求网络数据
                    requestData();
                } else {
                    sendErrorMessage(HANDLER_SHOW_ERROR, "没有网络");
                }
            }

        }
    });
}

//请求网络数据
public void requestData() {
//        http://c.m.163.com/nc/article/list/T1467284926140/0-20.html
//        http://c.m.163.com/nc/article/list/T1348647909107/0-20.html
    //确保当前没有处于联网刷新状态，以免重复发起请求
    if (!isConnectState) {
        //启动一个新的线程来执行网络请求操作。
        mThreadPool.execute(new Runnable() {
            @Override
            public void run() {
                //表示当前处于联网刷新状态，以防止重复请求
                isConnectState = true;
                //发起网络请求
```

```
                HttpHelper.get(mUrl, new HttpCallbackListener() {
//                    网络请求成功
                    @Override
                    public void onSuccess(String result) {
                        //解析返回的网络数据，并将解析结果存储在 mNewsListNormalBeanList 中
                        mNewsListNormalBeanList = DataParse.NewsList(result, tid);
                        //发送一条消息给主线程，通知主线程展示新闻数据（HANDLER_SHOW_NEWS）
                        if (mNewsListNormalBeanList != null) {
                            Message message = mHandler.obtainMessage();
                            message.what = HANDLER_SHOW_NEWS;
                            mHandler.sendMessage(message);
                            //保存最新刷新的时间戳
                            saveUpdateTime(tid, System.currentTimeMillis());
                            //将网络请求的结果缓存到本地
                            saveCache(mUrl, result);
                        }
                        isConnectState = false;
                    }

                    @Override
                    public void onError(Exception e) {
                        // 展示错误页面并尝试重新发出请求
                        LogUtils.e(TAG, "requestData" + e.toString());
                        //发送一条错误消息给主线程，通知主线程显示错误信息（HANDLER_SHOW_ERROR）
                        sendErrorMessage(HANDLER_SHOW_ERROR, e.toString());
                        //标记结束联网刷新状态：不管请求成功还是失败，最后都会将 isConnectState 标志重新设置为 false，表示联网刷新结束
                        isConnectState = false;
                    }});  } });}  }
```

```
    //显示新闻列表数据，并设置了点击事件处理逻
    @Override
    public void bindData() {
        //创建了一个名为 mNewsListAdapter 的自定义 NewsListAdapter 适配器，
        // 并将 mNewsListNormalBeanList 数据集合传递给它。
        // 这个适配器负责将新闻数据绑定到 RecyclerView 中。
        mNewsListAdapter = new NewsListAdapter(MyApplication.getContext(),
                (ArrayList<NewsListNormalBean>) mNewsListNormalBeanList);
        //将创建的适配器与 RecyclerView 关联起来，实现数据显示。
        mIRecyclerView.setIAdapter(mNewsListAdapter);
        // 设置 Item 点击跳转事件，当用户点击了列表中的某个条目时会触发
onItemClick
        mNewsListAdapter.setOnItemClickListener(new
NewsListAdapter.OnItemClickListener() {
            @Override
            public void onItemClick(View v, int position) {
                //获取点击的条目位置 position 对应的新闻数据
                NewsListNormalBean newsListNormalBean =
mNewsListNormalBeanList.get(position);
                String photosetID = newsListNormalBean.getPhotosetID();
                Intent intent;
                // newsListNormalBean 中是否包含 photosetID
                if (photosetID != null) {
                    //如果包含，表示这是一个图片新闻，
                    // 需要跳转到图片新闻详情页面。
                    intent = new Intent(getActivity(), PicDetailActivity.class);
                    String[] str = photosetID.split("\\|");
                    //  图片新闻文章所属的类目 id
                    String tid = str[0].substring(4);
                    // 图片新闻的文章 id 号
                    String setid = str[1];
                    intent.putExtra("TID", tid);
                    intent.putExtra("SETID", setid);
                    LogUtils.d(TAG, "onItemClick: photosetID:" + photosetID);
                } else {
```

```
                    //如果不是图片新闻，启动新闻详细展示页面，
                    intent = new Intent(getActivity(), NewsDetailActivity.class);
                    // 同时将新闻的唯一标识 DOCID 作为参数传递给目标页面。
                    intent.putExtra("DOCID", newsListNormalBean.getDocid());
                }
                //通过 getActivity().startActivity(intent) 启动相应的新闻详情页面。
                getActivity().startActivity(intent);
            }
        });

    }

    @Override
    public void initListener() {

        mIRecyclerView.setOnRefreshListener(new OnRefreshListener() {
            @Override
            public void onRefresh() {
                DownToRefresh();
            }
        });

        mIRecyclerView.setOnLoadMoreListener(new OnLoadMoreListener() {
            @Override
            public void onLoadMore() {
                if (mLoadMoreFooterView.canLoadMore() &&
mNewsListAdapter.getItemCount() > 0) {
                    PullUpToRefresh();
                }
            }
        });
    }

    // 下拉刷新
    public void DownToRefresh() {
```

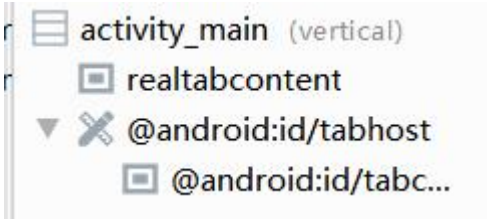
```
        //如果不在联网刷新状态，就可以执行下拉刷新操作
        if (!isConnectState) {
            //构建刷新请求 URL   0 表示刷新的起始位置
            mUrl = Api.CommonUrl + tid + "/" + 0 + Api.endUrl;
            mThreadPool.execute(new Runnable() {
                @Override
                public void run() {
                    isConnectState = true;//表示当前正在联网刷新
                    //发起 HTTP GET 请求 返回获得资源的字符串
                    HttpHelper.get(mUrl, new HttpCallbackListener() {
                        @Override
                        public void onSuccess(String result) {
                            if (result != null) {
                                Message message = mHandler.obtainMessage();
                                message.what =
HANDLER_SHOW_REFRESH_LOADMORE;
                                message.obj = result;
                                //在主线程中处理和显示刷新后的数据
                                mHandler.sendMessage(message);
                                //保存最后刷新时间，以便后续的缓存和时间戳判断
                                saveUpdateTime(tid, System.currentTimeMillis());
                                //缓存刷新后的新闻数据，以便后续的离线浏览。
                                saveCache(mUrl, result);
                            }
                        }

                        @Override
                        public void onError(Exception e) {
                            LogUtils.e(TAG, "requestData" + e.toString());
                            sendErrorMessage(HANDLER_SHOW_REFRESH_LOA
DMORE_ERRO, e.toString());
                        }
                    });
                }
            });
```

```
        }
    }

    // 上拉刷新
    public void PullUpToRefresh() {   …… ……    };

    /**
     * 上拉或下拉刷新之后更新 UI 界面
     */
    private void DataChange() {
        if (newlist != null) {
            isPullRefreshView();
            ToastUtils.showShort("数据已更新");
        } else {
            ToastUtils.showShort("数据请求失败");
        }
        mIRecyclerView.setRefreshing(false);
    }

……
```

二、程序入口（MainActivity）

前端：

使用 FrameLayout(帧布局)，一个底部标签栏导航的界面布局，底部标签栏位于底部， 上方是一个 FrameLayout，用于展示各个标签对应的 Fragment 内容。




```
<!--    总的来说，这段 XML 布局文件描述了一个底部标签栏导航的界面布局，底部标签栏位于底部，
上方是一个 FrameLayout，用于展示各个标签对应的 Fragment 内容。这种布局方式使得底部标签栏和内容区域能够共享屏幕高度。-->
```

```
<!--    LinearLayout 标签：这是根布局，设置了布局的宽度和高度都为
match_parent，使其填满父容器。
android:orientation="vertical"表示子视图按垂直方向排列。-->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="cn.bproject.neteasynews.activity.MainActivity">
    <!-- FrameLayout 标签：这个 FrameLayout 用于承载 Fragment 的内容。它的
宽度设置为 fill_parent，高度设置为 0dip 并使用权重分配空间。
    这样设置可以让底部标签栏上面的内容占据剩余的空间，使得底部标签栏和
内容能够平分屏幕高度。

    FrameLayout(帧布局)
    在屏幕上开辟出一块空白的区域,当我们往里面添加控件的时候,会默认把他
们放到这块区域的左上角,
    而这种布局方式却没有任何的定位方式
    -->
<!--    真正的内容视图,用于展示 Fragment-->
    <FrameLayout
        android:id="@+id/realtabcontent"
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_weight="1"
        android:background="@color/bg_color"
        />
    <!--    cn.bproject.neteasynews.widget.FragmentTabHost 标签：这是自定义的
FragmentTabHost 组件，
     用于实现底部标签栏。它的宽度设置为 fill_parent，高度设置为 wrap_content，
背景颜色为白色。-->
    <cn.bproject.neteasynews.widget.FragmentTabHost
        android:id="@android:id/tabhost"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@color/white">
        <!--     嵌套的 FrameLayout 标签：在 FragmentTabHost 中有一个嵌套的
FrameLayout，
        用于承载各个标签对应的 Fragment 内容，它的宽度和高度都设置为 0dp
并使用权重分配空间。-->
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:layout_weight="0" />
    </cn.bproject.neteasynews.widget.FragmentTabHost>
</LinearLayout>
```

后端主要代码及解释：

使用 FragmentTabHost 实现底部标签栏导航的 Android 应用程序主活动。
实现了一个具有底部标签栏导航功能的主活动。通过 FragmentTabHost 来管理标签和相关联的 Fragment，在标签切换时可以执行相应的逻辑。

```
public class MainActivity extends AppCompatActivity {
    private final String TAG = MainActivity.class.getSimpleName();

//     用于设置标签和关联的 Fragment
    private FragmentTabHost mTabHost;
    //     Layout 是一个用于加载布局的系统服务，就是实例化与 Layout XML 文
件对应的 View 对象，不能直接使用，
    //     需要通过 getLayoutInflater( )方法或 getSystemService( )方法来获得与当
前 Context 绑定的  LayoutInflater 实例
    private LayoutInflater mInflater;
    //底部标签栏内容的 List 集合
    private final List<BottomTab> mBottomTabs = new ArrayList<>(5);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initTab();

    }

    //    该方法用于初始化底部标签栏，首先创建了三个 BottomTab 对象，每个对象表示一个底部标签，
//        包括 Fragment 类、标签标题和图标资源。然后将这些标签对象添加到 mBottomTabs 列表中。
//        接着，通过 FragmentTabHost 来设置标签和关联的 Fragment。

    // 初始化底部标签栏
    private void initTab() {
        // 新闻标签
        BottomTab bottomTab_news = new BottomTab(NewsFragment.class,R.string.news_fragment,R.drawable.select_icon_news);
        ……
        mBottomTabs.add(bottomTab_news);
        ……
        // 初始化 mTabHost

        //LayoutInflacter 用法  1.获取 LayoutInflater 实例
        mInflater = LayoutInflater.from(this);
        //获取标签栏
        mTabHost = findViewById(android.R.id.tabhost);
        //获取标签栏和其 fragment
        // FragmentManager 类负责在应用的 fragment 上执行一些操作，如添加、移除或替换操作，以及将操作添加到返回堆栈。
        mTabHost.setup(this, getSupportFragmentManager(), R.id.realtabcontent);

        //mTabHost 获取内容和标签
```

```
//        将每个 BottomTab 对象转换为 TabHost.TabSpec 并添加到 mTabHost（一个 FragmentTabHost 实例）
        for (BottomTab bottomTab : mBottomTabs){
//            TabHost 相当于浏览器中浏览器分布的集合，而 Tabspec 则相当于浏览器中的每一个分页面。
//            在 Android 中，每一个 TabSpec 分布可以是一个组件，也可以是一个布局，然后将每一个分页装入 TabHost 中，
//            TabHost 即可将其中的每一个分页一并显示出来。

            //用于表示一个分页
            TabHost.TabSpec tabSpec =
mTabHost.newTabSpec(getString(bottomTab.getTitle()));

//            tabSpec.setIndicator()：使用 setIndicator()方法设置分页的指示器（标题和图标等）。
//                                    buildIndicator(bottomTab)返回一个视图对象，该视图将在分页选项卡上显示
            tabSpec.setIndicator(buildIndicator(bottomTab));

//            mTabHost.addTab()：通过 addTab()方法将创建的 TabSpec 对象添加到 mTabHost 中。
//            第一个参数是分页的标签，第二个参数是分页对应的 Fragment 类，第三个参数是传递给 Fragment 的参数，这里设置为 null。
            mTabHost.addTab(tabSpec, bottomTab.getFragment(),null);

        }

        mTabHost.setOnTabChangedListener(new TabHost.OnTabChangeListener()
{
            @Override
            public void onTabChanged(String tabId) {

                LogUtils.d(TAG, "onTabChanged:
mTabHost.setOnTabChangedListener" + R.string.news_fragment);
```

```
            }
        });
//          用于设置选项卡之间的分割线显示。通过 getTabWidget()方法获取 mTabHost 的标签部件，
//          然后使用 setShowDividers()方法将分割线的显示设置为 LinearLayout.SHOW_DIVIDER_NONE，即不显示分割线。
        mTabHost.getTabWidget().setShowDividers(LinearLayout.SHOW_DIVIDER_NONE);
//          设置当前显示的选项卡为索引为 0 的选项卡。它将默认显示第一个选项卡。
        mTabHost.setCurrentTab(0);

    }
    //buildIndicator 方法：用于构建底部标签的视图，从 tab_indicator.xml 布局文件中加载视图，设置图标和文本，然后返回构建好的视图。

    // 设置底部 tab 的图片和文字     Indicator：标志
    private View buildIndicator(BottomTab bottomTab){

        View view = mInflater.inflate(R.layout.tab_indicator, null);
        ImageView img = view.findViewById(R.id.icon_tab);
        TextView text = view.findViewById(R.id.txt_indicator);

        img.setBackgroundResource(bottomTab.getIcon());
        text.setText(bottomTab.getTitle());

        return  view;
    }


    //Intent 双向传值，这里根据返回的结果码和附加数据，接收值，用来切换新闻模块的频道
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
```

```
        //获取当前所显示的选项卡的标签。
        String tag =   mTabHost.getCurrentTabTag();
        if (resultCode == 789){
            Bundle bundle = data.getExtras();
            //从附加数据中获取名为 "NewTabPostion" 的整数数据，这表示新的选项卡位置。
            int tabPosition = bundle.getInt("NewTabPostion");
            //通过标签找到当前显示的 NewsFragment 片段。
            NewsFragment newsFragment = (NewsFragment) getSupportFragmentManager().findFragmentByTag(tag);
            //将新的选项卡位置传递给它，以便它可以切换到相应的频道。
            newsFragment.setCurrentChannel(tabPosition);
            //使片段可以更新其内容以反映新的频道配置。
            newsFragment.notifyChannelChange();
        }
    }
}
```

三、共用的 Fragment 和 layout

新闻，图片，视频模块共有 fragment 或多个模块用到的 layout(xml)文件

toolbar_page.xml

merge 并不是一个 ViewGroup，也不是一个 View，它相当于声明了一些视图，等待被添加。

merge 标签被添加到 A 容器下，那么 merge 下的所有视图将被添加到 A 容器下

```
public abstract class BaseFragment extends Fragment implements DefineView {
    public final int HANDLER_SHOW_NEWS = 11;
    public final int HANDLER_SHOW_ERROR = 12;
    public final int HANDLER_SHOW_REFRESH_LOADMORE = 13;
    public final int HANDLER_SHOW_REFRESH_LOADMORE_ERRO = 15;


    private final String TAG = BaseFragment.class.getSimpleName();

    /**
     * 设置 toolbar 标题居中，没有返回键
     * @param view
     * @param id       toolbar 的 id
     * @param titleId     textView 的 id
     * @param titleString     textView 设置的文字
     * @return    返回 toolbar
     */
    public Toolbar initToolbar(View view, int id, int titleId, int titleString) {
        Toolbar toolbar = (Toolbar) view.findViewById(id);
//          toolbar.setTitle("");
        TextView textView = (TextView) view.findViewById(titleId);
        textView.setText(titleString);
        AppCompatActivity activity = (AppCompatActivity) getActivity();
        activity.setSupportActionBar(toolbar);
        android.support.v7.app.ActionBar actionBar =
activity.getSupportActionBar();
        if (actionBar != null){
            actionBar.setDisplayHomeAsUpEnabled(false);
            actionBar.setDisplayShowTitleEnabled(false);
         }
        return toolbar;
     }

    /**
```

```
     *
     * @param key
     * @return   返回 true 表示是最新的保存的，返回 false 表示保存已经超过 3 个小时
     */
    public boolean isLastNews(String key) {
        long threeHour = 3 * 60 * 60 * 1000;
        long currentTime = System.currentTimeMillis();
        long saveTime = getUpdateTime(key, currentTime);
        // 判断保存缓存的时间与现在的时间是否超过 3 小时，没有超过就读取缓存
        long ll = currentTime - saveTime;

        if (ll <= threeHour) {
            LogUtils.d(TAG, "saveTime   :   " + saveTime + "    ll:   " + ll + " ll < threeHour " + (ll < threeHour));
            return true;
        } else {
            LogUtils.d(TAG, "saveTime   :   " + saveTime + "    ll:   " + ll + " ll > threeHour " + (ll > threeHour));
            return false;
        }
    }

    // 保存缓存
    public void saveCache(String url, String content){
        if (LocalCacheUtils.hasCacheFile(url)) {
            // 清除之前的缓存
            LocalCacheUtils.removeCache(url);
        }
        // 保存缓存
        LocalCacheUtils.setDiskLruCache(url, content);
    }

    // 设置保存缓存的时间
```

```
    public static void saveUpdateTime(String key, long value) {
        PrefUtils.setLong(MyApplication.getContext(), "save_time", key, value);
    }

    // 获取保存缓存的时间
    public static long getUpdateTime(String key, long defValue) {
        long saveTime = PrefUtils.getLong(MyApplication.getContext(), "save_time",
key, defValue);
        return saveTime;
    }
}
```

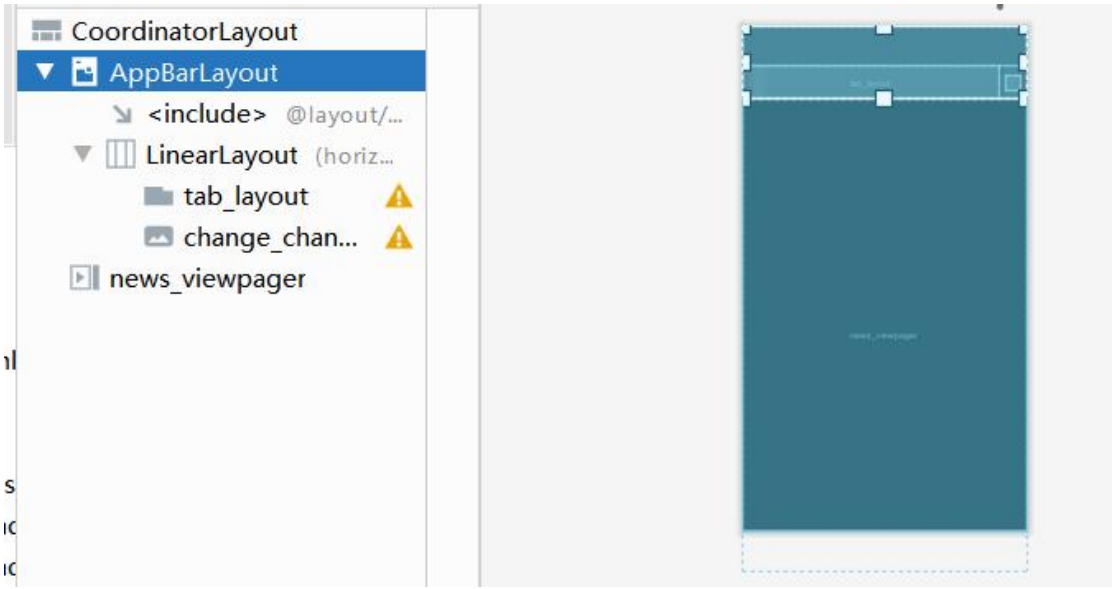
Tablayout_page.xml

LinearLayout：新闻、图片、视频三大模块统一使用 taglayout + viewpager 作为滑动页面

tab-layout:显示各种频道，点击切换

Change_channel:进入频道管理界面的入口

news_viewpager:各个频道的新闻内容

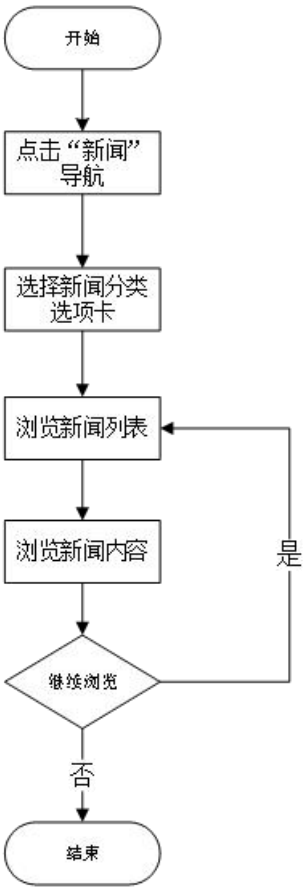


四、新闻模块

通过网络请求获取新闻数据在数据获取模块

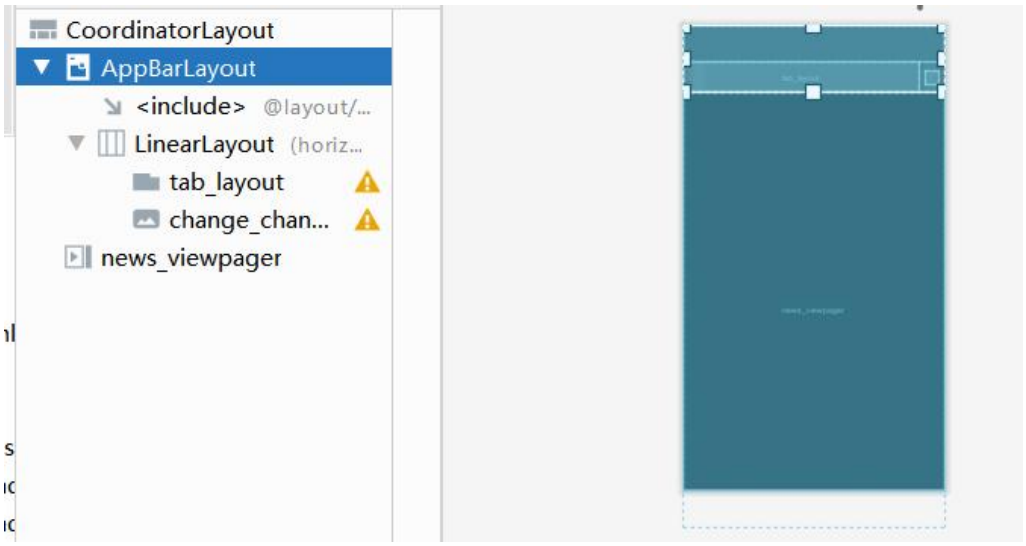
新闻浏览流程：

点击新闻导航按钮，可以进入新闻浏览列表，可以选择对应的新闻分类进行浏览，点击右上方的“+”号，可以进入分类管理界面，添加自己喜欢的新闻分类，实际上就是一个新闻关键词，也可以选择推荐的新闻分类，回到新闻列表，点击新闻列表中的内容，就可以进行新闻图文浏览。



（一）进入新闻模块

前端：



后端主要步骤以及代码详解：

大体步骤为：

1. 设置 toolbar，显示当前是哪个模块
2. 进行 TabLayout 和 ViewPager 的关联，并绑定数据
3. 设置点击事件，点击频道(TabLayout 内的标签)后切换 ViewPager(不同频道显示的内容)

具体代码及注释：

```
public class NewsFragment extends BaseFragment {

    private final String TAG = NewsFragment.class.getSimpleName();

    private TabLayout mTabLayout;//显示标签，切换不同的新闻频道
    private ViewPager mNewsViewpager;//显示不同频道对应的内容页面
    private View mView;//Fragment 的根视图
    private FixedPagerAdapter fixedPagerAdapter;//ViewPager 的适配器，用于管理不同频道的内容 Fragment
    private List<BaseFragment> fragments;//存储各个频道对应的内容 Fragment
```

```
    private List<ProjectChannelBean> myChannelList;//用户设置的新闻频道列表
    private List<ProjectChannelBean> moreChannelList;//更多的新闻频道列表
    private ImageButton mChange_channel;//切换频道的按钮。
    private int tabPosition;//  当前新闻频道的位置
    private SharedPreferences sharedPreferences;//用于存储应用设置信息。
    private ListDataSave listDataSave;//保存频道数据的工具类。 将 List 集合转为 json 数据保存在 sharedPreferences 的工具类
    private boolean isFirst;//标记是否为第一次进入应用。
    private BaseFragment baseFragment;//用于存储创建的 Fragment 对象。

    @Nullable
    @Override
//     在片段被创建时调用，加载片段的布局。使用布局文件 tablayout_pager.xml 来创建视图。
    //LayoutInflater inflater：用于从布局资源文件创建视图的工具。
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.tablayout_pager, container, false);
        return mView;
    }

    @Override
    //初始化视图中的各个控件和数据
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        initView();
    }

    @Override
    public void initView() {
        mTabLayout = mView.findViewById(tab_layout);//新闻标签
        mNewsViewpager = mView.findViewById(R.id.news_viewpager);//新闻内容
        mChange_channel = mView.findViewById(R.id.change_channel);//改变频道
```

```
        Toolbar myToolbar = initToolbar(mView, R.id.my_toolbar, R.id.toolbar_title,
R.string.news_home);
        //初始化，设置 TabLayout 和 ViewPager 的关联，并绑定数据
        initValidata();
        //点击标签，更换频道内容
        initListener();
    }

    @Override
    //于初始化一些数据和实例，设置 TabLayout 和 ViewPager 的关联，并绑定
数据
    public void initValidata() {
        //只有当前应用可以访问，获取的 Setting 共选项
        sharedPreferences = getActivity().getSharedPreferences("Setting",
Context.MODE_PRIVATE);
        //保存频道数据为 Json，取名为 channel
        listDataSave = new ListDataSave(getActivity(), "channel");
        //用于存储各个频道对应的内容片段
        fragments = new ArrayList<BaseFragment>();
        //管理 ViewPager 中的内容的适配器
        fixedPagerAdapter = new FixedPagerAdapter(getChildFragmentManager());
        //将 TabLayout 与 ViewPager 进行关联，实现选项卡与页面内容的同步
切换。
        mTabLayout.setupWithViewPager(mNewsViewpager);
        //将数据绑定到 ViewPager 和适配器上，实现不同频道内容的显示
        bindData();
    }

    @Override
    //点击标签，更换频道内容
    public void initListener() {
        mTabLayout.addOnTabSelectedListener(new
TabLayout.OnTabSelectedListener() {
            @Override
            public void onTabSelected(TabLayout.Tab tab) {
```

```
                tabPosition = tab.getPosition();
……
        });
        //切换频道
        mChange_channel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getActivity(),
ChannelManagerActivity.class);
                //传递的数据为当前频道位置
                intent.putExtra("TABPOSITION", tabPosition);
                startActivityForResult(intent, 999);
            }
        });
    }

    @Override
    public void bindData() {
        //获取所有频道
        getDataFromSharedPreference();
        //存储频道列表
        fixedPagerAdapter.setChannelBean(myChannelList);
        //存储各个频道内容
        fixedPagerAdapter.setFragments(fragments);
        //给文章的显示设置适配器
        mNewsViewpager.setAdapter(fixedPagerAdapter);
    }

    /**
     * 判断是否第一次进入程序
     * 如果第一次进入，直接获取设置好的频道
     * 如果不是第一次进入，则从 sharedPrefered 中获取设置好的频道
     */
private void getDataFromSharedPreference() {
        isFirst = sharedPreferences.getBoolean("isFirst", true);
```

```
        if (isFirst) {
            //获取频道数据列表。
            myChannelList = CategoryDataUtils.getChannelCategoryBeans();
            //获取更多频道数据。
            moreChannelList = getMoreChannelFromAsset();
            //设置频道数据的类型。
            myChannelList = setType(myChannelList);
            moreChannelList = setType(moreChannelList);
            //将设置好的频道数据保存到共享偏好设置中，并将 "isFirst" 设置为
false。
            listDataSave.setDataList("myChannel", myChannelList);
            listDataSave.setDataList("moreChannel", moreChannelList);
            SharedPreferences.Editor edit = sharedPreferences.edit();
            edit.putBoolean("isFirst", false);
            edit.commit();
        } else {
            //不是第一次进入程序,从共享偏好设置中获取之前保存的频道数据列
表。
            myChannelList = listDataSave.getDataList("myChannel",
ProjectChannelBean.class);
        }
        //清空 fragments 列表，用于重新填充内容片段列表
        fragments.clear();
        //为每个频道创建对应的内容片段，并将内容片段添加到 fragments 列表
中。
        for (int i = 0; i < myChannelList.size(); i++) {
            baseFragment =
NewsListFragment.newInstance(myChannelList.get(i).getTid());
            fragments.add(baseFragment);
        }
        //根据 myChannelList 的大小判断是否启用 TabLayout 的固定模式
（TabLayout.MODE_FIXED）
        // 或滚动模式（TabLayout.MODE_SCROLLABLE）。
        if (myChannelList.size() <= 4) {
            mTabLayout.setTabMode(TabLayout.MODE_FIXED);
```

```
        } else {
            mTabLayout.setTabMode(TabLayout.MODE_SCROLLABLE);
        }
    }

    /**
     * 在 ManiActivty 中被调用，当从 ChanelActivity 返回时设置当前 tab 的位置
     * @param tabPosition
     */
    public void setCurrentChannel(int tabPosition) {
        mNewsViewpager.setCurrentItem(tabPosition);
        mTabLayout.setScrollPosition(tabPosition, 1, true);
    }

    /**
     * 在 myChannelList 发生改变的时候更新 ui，在 MainActivity 调用
     */
    public void notifyChannelChange() {
        getDataFromSharedPreference();
        fixedPagerAdapter.setChannelBean(myChannelList);

        fixedPagerAdapter.setFragments(fragments);
        fixedPagerAdapter.notifyDataSetChanged();

    }

    private List<ProjectChannelBean> setType(List<ProjectChannelBean> list) {
        Iterator<ProjectChannelBean> iterator = list.iterator();
        while (iterator.hasNext()) {
            ProjectChannelBean channelBean = iterator.next();
            channelBean.setTabType(APPConst.ITEM_EDIT);
        }
        return list;
    }
```

```
    /**
     * 从 Asset 目录中读取更多频道
     *
     * @return
     */
    public List<ProjectChannelBean> getMoreChannelFromAsset() {
        String moreChannel = IOUtils.readFromFile("projectChannel.txt");
        List<ProjectChannelBean> projectChannelBeanList = new ArrayList<>();
        JsonArray array = new JsonParser().parse(moreChannel).getAsJsonArray();
        for (final JsonElement elem : array) {
            projectChannelBeanList.add(new Gson().fromJson(elem,
ProjectChannelBean.class));
        }
        return projectChannelBeanList;
    }
}
```

文章内容适配器：

处理内容片段的实例化、销毁、标题获取等操作，并允许在不同频道之间进行切换。
通过设置频道数据和内容片段列表，适配器可以根据位置动态获取对应的内容片段。
适配器对应的内容获取在内容获取模块

```
public class FixedPagerAdapter extends FragmentStatePagerAdapter {
    //存储频道数据的列表。
    private List<ProjectChannelBean> channelBeanList;
    //管理片段
    private FragmentManager fm;
    //存储不同频道对应的内容片段列表
    private List<BaseFragment> fragments;

    //初始化适配器
    public FixedPagerAdapter(FragmentManager fm) {
```

```
        super(fm);
        this.fm = fm;
    }
    //设置频道数据列表
    public void setChannelBean(List<ProjectChannelBean> newsBeans) {
        this.channelBeanList = newsBeans;
    }
    //设置内容片段列表
    public void setFragments(List<BaseFragment> fragments) {
        this.fragments = fragments;
    }
    //根据位置获取对应的内容片段
    @Override
    public BaseFragment getItem(int position) {
        return fragments.get(position);
    }
    //返回内容片段的数量
    @Override
    public int getCount() {
        return fragments.size();
    }
    //实例化内容片段
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        BaseFragment fragment = null;
        try {
            removeFragment(container, position);
            fragment = (BaseFragment) super.instantiateItem(container, position);
        } catch (Exception e) {

        }
        return fragment;
    }
    //移除指定位置的片段
    private void removeFragment(ViewGroup container,int index) {
```

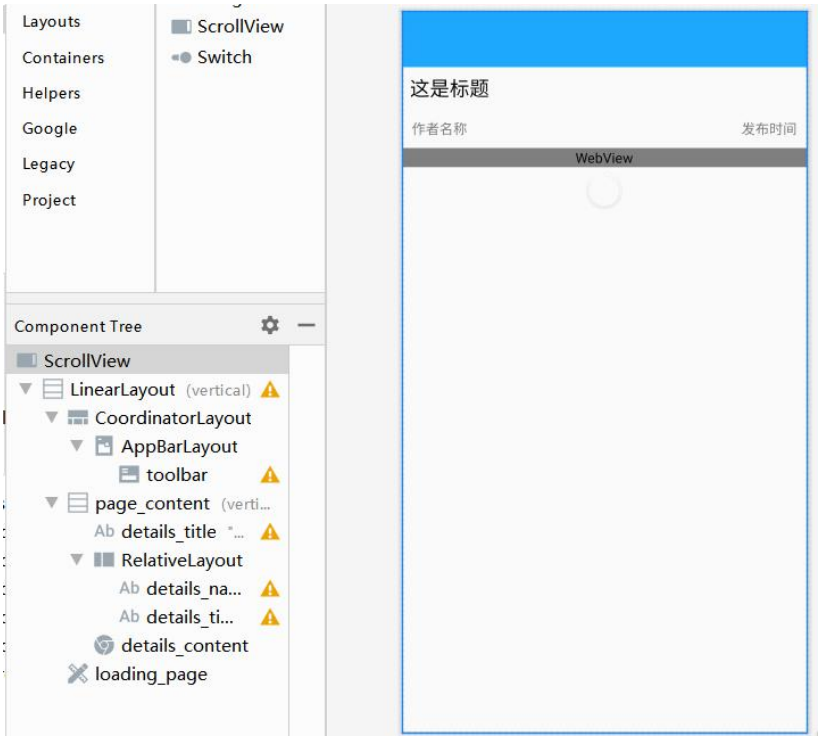
```
        String tag = getFragmentTag(container.getId(), index);
        Fragment fragment = fm.findFragmentByTag(tag);
        if (fragment == null)
            return;
        FragmentTransaction ft = fm.beginTransaction();
        ft.remove(fragment);
        ft.commit();
        ft = null;
        fm.executePendingTransactions();
    }
    //根据视图 ID 和索引获取片段的标签
    private String getFragmentTag(int viewId, int index) {
        try {
//            通过 Class 知道某个类中有多少方法，有多少字段，每个字段叫什么名字，
//            每个字段的类型是什么，每个方法的方法名是什么，某个方法有几个参数
            Class<FragmentPagerAdapter> cls = FragmentPagerAdapter.class;
            Class<?>[] parameterTypes = { int.class, long.class };
            Method method = cls.getDeclaredMethod("makeFragmentName",
                    parameterTypes);
            method.setAccessible(true);
            String tag = (String) method.invoke(this, viewId, index);
            return tag;
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
}
```

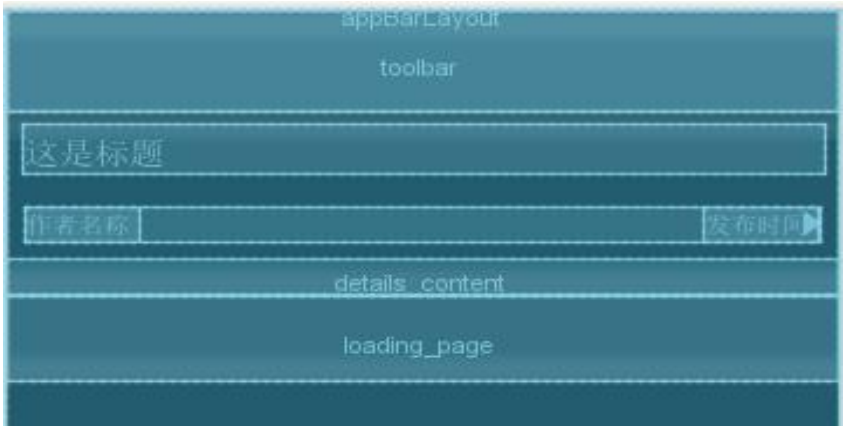
（二）点击单个新闻显示的具体内容

点击进入单个新闻后，会加载点击新闻的标题，发布者，发布时间，以及其主要内容(点击跳转在数据获取模块）

前端：

外层的 ScrollView 元素用于支持滚动，以便在内容过长时进行滚动查看。
AppBarLayout 实现了一个折叠的应用程序栏
CoordinatorLayout 可以协调各个子视图
webview 通过加载 html 文件来展示新闻具体内容
Loading_page 是自定义的加载新闻内容布局





后端详细主要代码解释：

主要包括初始化视图、数据，发起网络请求以获取详细数据，处理 UI 更新。

```
public class NewsDetailActivity extends BaseActivity implements DefineView {
    private final String TAG = NewsDetailActivity.class.getSimpleName();
    //      显示新闻标题、作者和发布时间
    private TextView details_title, details_name, details_time;
    //上下文      通过 servlet 绑定数据使用  作为中间的通道让 Servlet  和 Web 容器进行交互
    private Context mContext;
    //用于显示新闻内容      WebView  类是  Android  的  View  类的扩展，让网页显示为  Activity  布局的一部分
    private WebView mWebView;
    private ThreadManager.ThreadPool mThreadPool;    // 线程池
    //WebView 的配置。
    private WebSettings mWebSettings;
    //存储偏好设置
    private SharedPreferences sharedPreferences;
    //新闻的唯一 ID。
    private String mDocid;
    //存储获取的新闻详情数据。
    private NewsDetailBean mNewsDetailBeen;
    //包裹内容的布局
    private LinearLayout mPage_content;
//      加载页面部件
    private LoadingPage mLoadingPage;
    //处理 UI 更新    Handler  主要用于在后台线程中与 UI 线程进行通信，以便在 UI 线程上执行一些任务。
    private final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
```

```
            //绑定数据到 UI 元素上
            bindData();
            //显示新闻页面
            showNewsPage();
        }
    };

    //在活动创建时调用，初始化视图、数据和监听器
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_newsdetail);
        mContext = this;
        Intent intent = getIntent();
        mDocid = intent.getStringExtra("DOCID");
        initView();
        initValidata();
        initListener();
    }


    //初始化视图，包括设置 Toolbar、获取相关视图和部件，展示加载页面
    @Override
    public void initView() {
        initToolbar();
        sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(mContext);
        mPage_content = findViewById(R.id.page_content);
        mLoadingPage = findViewById(R.id.loading_page);
        details_title = this.findViewById(R.id.details_title);
        // 设置标题加粗
        TextPaint tp = details_title.getPaint();
        tp.setFakeBoldText(true);
        details_name = this.findViewById(R.id.details_name);
        details_time = this.findViewById(R.id.details_time);
```

```
        mWebView = this.findViewById(details_content);

        showLoadingPage();
    }
    //初始化工具栏，包括设置工具栏的标题和返回按钮。
    private void initToolbar(){
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
            actionBar.setHomeAsUpIndicator(R.drawable.icon_back);
        }
    }


    //初始化数据，设置 WebView 的配置，添加 Javascript 接口对象，发起网络请求获取新闻详情。
    @Override
    public void initValidata() {
        setWebView();
        // 将设置好的 JavaScriptInterface 对象传入，第二个参数则是为这个对象设置名称（可随意）
        mWebView.addJavascriptInterface(new JavaScriptInterface(), "androidMethod");
        requestData();
    }
    /**
     * 设置 WebView 相关配置
     * 包括自适应屏幕、支持缩放和 JavaScript 等
     */
    private void setWebView(){
        mWebSettings = mWebView.getSettings();
        //自适应屏幕
```

```
        mWebSettings.setLayoutAlgorithm(WebSettings.LayoutAlgorithm.SINGLE_
COLUMN);
        mWebSettings.setLoadWithOverviewMode(true); // 缩放至屏幕的大小
        // 打开页面时， 自适应屏幕
        mWebSettings.setUseWideViewPort(true); //将图片调整到适合 webview 的
大小
        mWebSettings.setSupportZoom(true); //支持缩放
        mWebSettings.setJavaScriptEnabled(true);  //开启 javascript
        mWebSettings.setDomStorageEnabled(true);  //开启 DOM
        mWebSettings.setDefaultTextEncodingName("utf-8"); //设置编码
        // // web 页面处理
        mWebSettings.setAllowFileAccess(true);// 支持文件流

        //提高网页加载速度，暂时阻塞图片加载，然后网页加载好了，再进行加
载图片
        mWebSettings.setBlockNetworkImage(true);
        //开启缓存机制
        mWebSettings.setAppCacheEnabled(true);
        setTextSize();
        //设置 webview
        mWebView.setWebChromeClient(new MyWebChromeClient());
        mWebView.setWebViewClient(new MyWebViewClient());
    }
    //发起网络请求，获取新闻详情数据。
    private void requestData(){
        // 创建线程池,用于执行网络请求的任务
        mThreadPool = ThreadManager.getThreadPool();
        mThreadPool.execute(new Runnable() {
            @Override
            public void run() {
                //通过拼接 mDocid 到请求 URL 中，构建了获取新闻详情的完整
URL 地址 url
                String url = Api.DetailUrl + mDocid + Api.endDetailUrl;
                Log.d(TAG, "文章 url 为: " + url);
                //发起 HTTP GET 请求，该请求会异步执行
```

```
                HttpHelper.get(url, new HttpCallbackListener() {
                    //在请求的回调函数中，如果请求成功（onSuccess 方法），则会解析获取到的结果字符串 result，
                    // 并将解析后的数据存储到 mNewsDetailBeen 对象中。
                    @Override
                    public void onSuccess(String result) {
                        mNewsDetailBeen = DataParse.NewsDetail(result, mDocid);
                        handler.sendMessage(handler.obtainMessage());
                    }

                    @Override
                    public void onError(final Exception e) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                Toast.makeText(mContext, e.toString(), Toast.LENGTH_LONG).show();
                                showErroPage();
                            }
                        });
                    }
                });
            }
        });
    }

    @Override
    public void initListener() {

    }

    @Override
    public void bindData() {
        //数据获取成功后
        if (mNewsDetailBeen != null) {
```

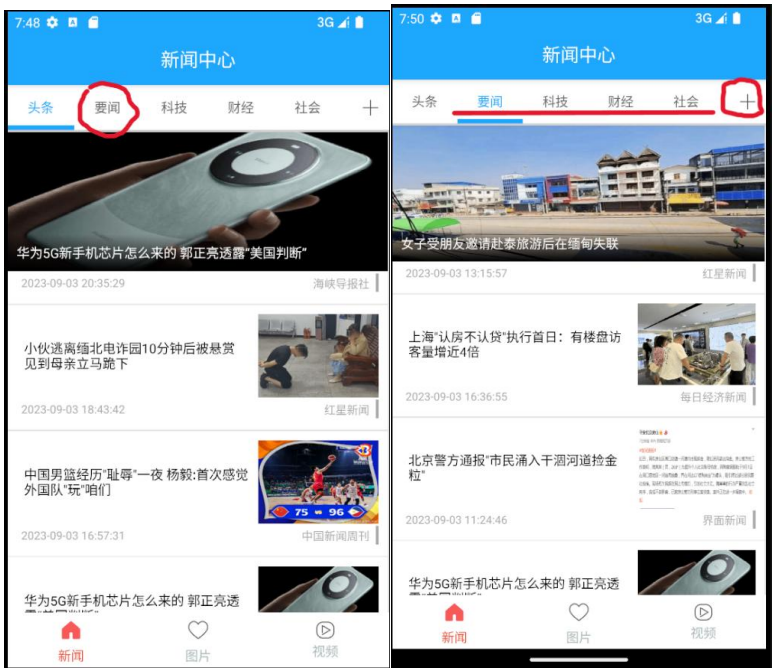
```
            //对新闻详情进行处理
            changeNewsDetail(mNewsDetailBeen);
            String body = mNewsDetailBeen.getBody();
            // 使用 css 样式的方式设置图片大小
            //用于设置图片的大小、页边距和字体大小等。这样可以确保新闻详情在 WebView 中以可读的方式显示
            String css = "<style type=\"text/css\"> img {" +
                    "width:100%;" +
                    "height:auto;" +
                    "}" +
                    "body {" +
                    "margin-right:15px;" +
                    "margin-left:15px;" +
                    "margin-top:15px;" +
                    "font-size:24px;" +
                    "}" +
                    "</style>";
            String html = "<html><header>" + css + "</header><body>" + body + "</body></html>";
            Log.d(TAG, "html: " + html);
            String title = mNewsDetailBeen.getTitle();
            String ptime = mNewsDetailBeen.getPtime();
            String source = mNewsDetailBeen.getSource();

            details_title.setText(title);
            details_name.setText(source);
            details_time.setText(ptime);
            //details_content.loadData(articleBean.getContext(),"text/html","UTF-8");

            //加载 HTML 内容到 WebView 中，以便显示新闻详情的正文内容。
            mWebView.loadDataWithBaseURL(null, html, "text/html", "UTF-8", "");
        } else{
            showEmptyPage();
        }
```

```
}
```

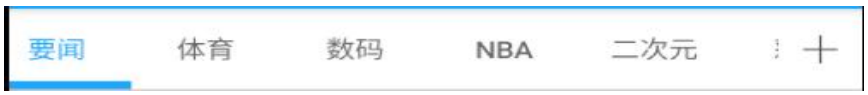
（4）系统测试

新闻与频道模块

1. 切换频道与频道管理



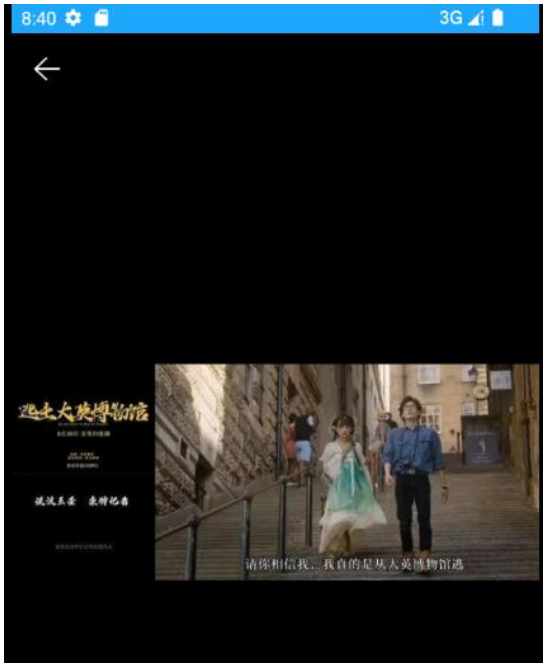
删除蓝色框内频道，添加彩票频道后



2. 查看新闻详情

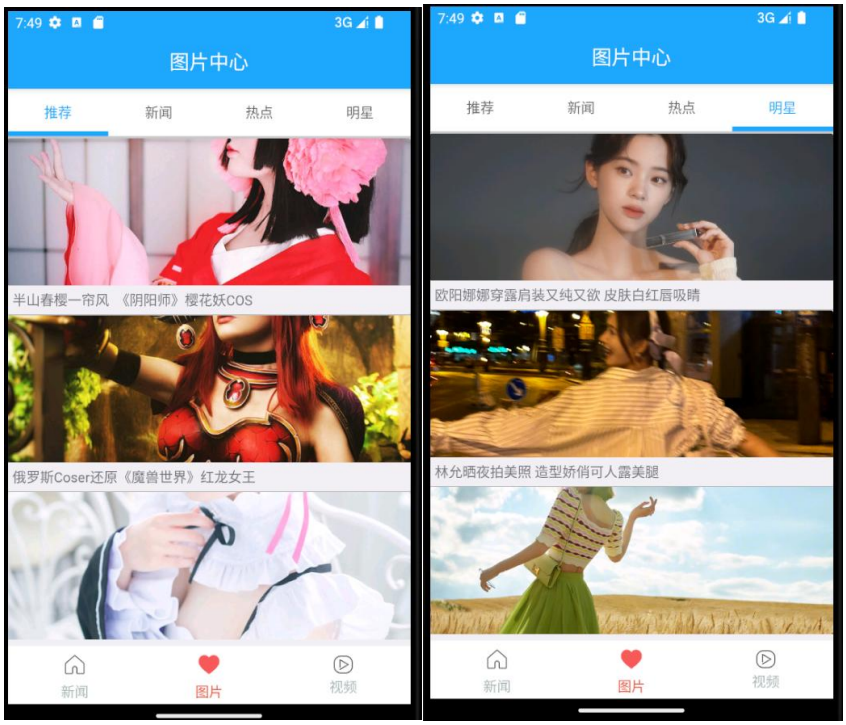
点击此新闻后显示详细新闻信息

点击新闻内图片



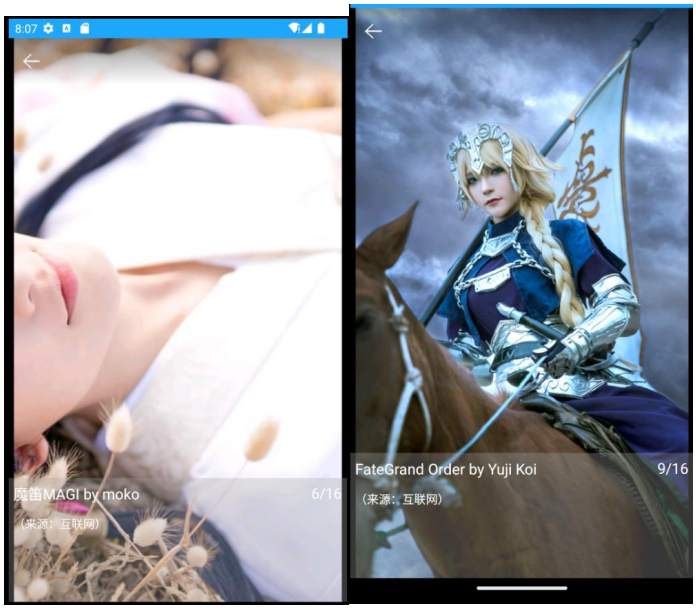
图片模块

1. 图片集显示



2. 查看图片

点击图片集封面后显示，滑动显示下一张。



视频模块


8:16
推荐视频
9月4日，三大消息值得关注
original
2023-09-04 11:09:40
朝鲜长公主金敬姬地位有多高？敢对金日成说不，金...
朝鲜初代长公主
金敬姬地位有多高
original
2023-09-01 17:58:23
我南京退休，收入和缴费比外省战友高，养老金却低...
因为谈起来就感觉脸红着慌
original
2023-09-01 17:29:39
新闻
图片
视频

（5）设计总结（设计遇到的问题和解决思路等）

这次课程设计在实现过程中十分困难，但同样收获满满。

一开始我准备做第一个课设题目，语音识别和图片识别，我申请了百度的AI 开放平台的权限，看了半天说明文档，下了一堆 API，最后没搞出来，老是API 调用错误，莫名其妙报错，之后在网上搜不到相关的教学视频，于是我只能转战第三个题目，新闻模块。开始想着怎么用课堂上的内容实现这次的设计，由于时间久远，我由看了一便课件，这个时候有了大概的想法，但涉及很多不会的知识。于是我找了很多视频教学，又搭建了环境，找了新闻项目的简易脚手架，开始不断完善此次课设。

在此之间问题遇到了很多很多，比如看不懂找到的项目脚手架，只能一点一点的百度还有用 chatgpt 和 CodeGeeX 搜，逐渐大概了前端上 FrameLayout 和FragmentTabHost 的使用，还有很多课件上没有的 UI 组件(如 IRecyclerView，还有自定义加载界面前端等等)来满足课设的大致排版。还有怎么抓取网易新闻 api，再存储网易新闻网站上各个频道类型新闻的 Url 然后再包装成 Api，用于在后端线程中读取并解析显示数据。期间，免不了在网上搜索各种的对我来说很新的知识，什么线程，Handler 用法，保存和使用缓存，前端各种东西的用法，怎么把网上的东西用在自己的项目中。

同时我也明白了基础的重要性，我在学安卓这门课的时候，java 其实已经忘得差不多了。由于这次两个课设都用的是 java，我提前花了一个星期又熟悉了一边java，在这次课程设计中，起到了很大的作用，在看课件的时候，有了新的理解，发现自己当初由于 java 基础知识不牢固，在上课的时候很多东西都没有能够理解，很多都是死记硬背的。在又复习了一边基础，安卓中的很多东西，能看出个大概为什么有这种效果，怎么实现的，而不是只能死记硬背。

同时，本次课设也用到了很多网络方面的知识，比如进程，线程，TID，UID，URL，HTTP/HTTPS，TCP/IP 网络相关知识，让我对下学期的计算机网络学习有了很大的兴趣。

总之，这是一次让我受益匪浅的课程涉及，他让我懂得了怎么为了达成项目要求，全方位的寻找组合能够获取的资源，同时，也让我知道了基础的重要性。