2024 年夏季 Java 小学期大作业报告

罗雪蘅 2023010877 计 38 经 32

1.代码结构

一．活动

代码共有 7 个不同的活动。

MainActivity 用于展示主界面，包括侧滑抽屉（可跳转历史列表、收藏列表、设置类别界面）、主工具栏（包括跳转搜索界面的按钮）、分类滑块（有动画）和新闻列表(展示新闻标题、来源、时间，并加载第一张图片（如果有）作为封面，点击可跳转新闻详情，顶部下拉刷新，底部上拉加载更多新闻)。

NewsDetailsActivity 用于展示新闻详情界面，加载新闻标题、来源、时间、新闻文字内容、大模型生成的摘要、新闻图片内容（如果有）、新闻视频（如果有），新闻详情界面上方有星型按钮（点击可添加收藏，再次点击取消收藏）和返回栏（点击图标可返回前一个界面）。

HistoryListActivity 用于展示历史列表，类似新闻列表，可跳转新闻详情，点击返回栏图标可返回主界面。

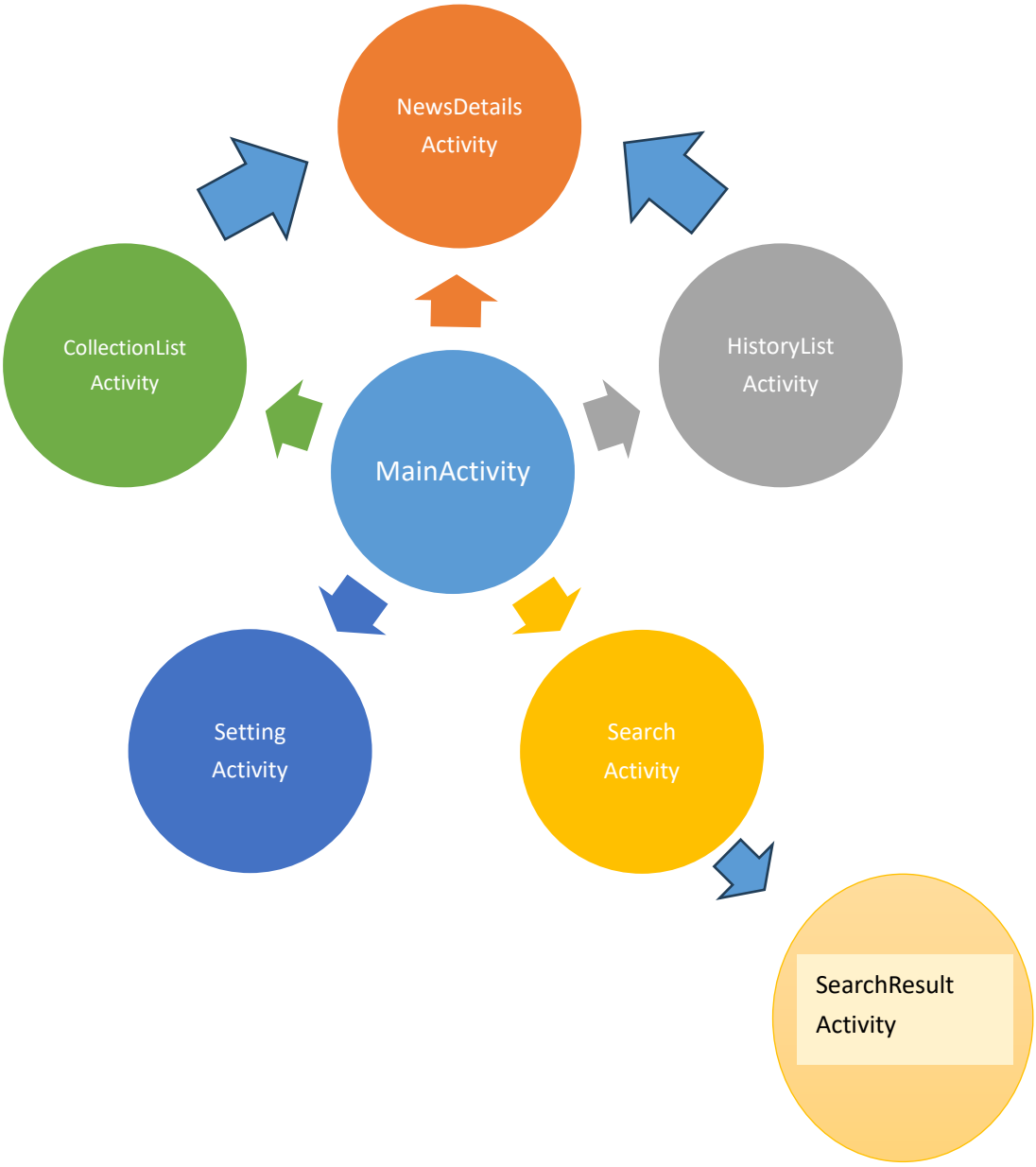
CollectionListActivity 用于展示历史列表，类似新闻列表，可跳转新闻详情，点击返回栏图标可返回主界面。

SettingActivity 用于展示设置类别的界面，包括十二个按钮（科技、社会、体育、娱乐、汽车、教育、文化、健康、军事、财经、其它、全部），每个按钮点击变红，再次点击变白，且初始颜色保留用户上次设置的状态（首次设置前默认类别全选）。页面右上角有一个确定按钮，点击表示提交此次类别修改（提交时类别按钮为红代表想要这个分类，为白表示不要），并自动跳转主界面。

SearchActivity 用于展示搜索界面，包括关键字输入框、开始日期选择按钮、结束日期选择按钮、类别输入框，点击右上角搜索按钮可跳转搜索结果界面，点击返回栏图标可返回主界面。

SearchResultActivity 用于展示搜索结果界面，展示所有符合用户搜索条件的新闻，类似新闻列表，点击返回栏图标可返回搜索界面。

活动之间跳转关系如下图所示：



二．实体 entity

NewsInfo 定义了从 url 爬取的新闻的具体格式，包括在展示新闻列表、详情时需要用到的 title,publisher,Publishtime,image,video 等内容。

HistoryInfo 定义了数据库 HistoryDb 中存储数据的格式。

CollectionInfo 定义了数据库 CollectionDb 中存储数据的格式。

PreferenceInfo 定义了数据库 PreferenceDb 中存储数据的格式。

AbstractInfo 定义了数据库 AbstractDb 中存储数据的格式。

三．DatabaseHelper

HistoryDbHelper 是数据库 HistoryDb 的 DatabaseHelper,创建了 History_table,并定义了增添数据、查找数据是否在其内、查找数据的方法。其中 HistoryDb 存储用户历史浏览的新闻信息，存储的信息包括历史数据 id，新闻 id，以及 NewsInfo 类新闻转化成的 json 文件类（携带新闻全部信息）。

CollectionDbHelper 是数据库 CollectionDb 的 DatabaseHelper,创建了 Collection_table,并定义了增添数据、删除数据、查找数据是否在其内、查找数据的方法。其中 CollectionDb 存储用户收藏的新闻信息，存储的信息包括收藏数据 id，新闻 id，以及 NewsInfo 类新闻转化成的 json 文件类（携带新闻全部信息）。

PreferenceDbHelper 是数据库 PreferenceDb 的 DatabaseHelper, 创建了 Preference_table, 并定义了增添数据、删除数据、查找数据是否在其内、查找数据的方法。 其中 PreferenceDb 存储用户编辑的类别信息，存储的信息包括类别数据 id，以及代表类别的字符串（如：“科技”）。 特别需要指出地，由于初始默认用户希望看到所有类别的新闻，所以只有当这个类别没有出现在数据库中时才显示这个类别的新闻。

AbstractDbHelper 是数据库 AbstractDb 的 DatabaseHelper,创建了 abstract_table, 并

定义了增添数据、查找数据是否在其内、查找数据的方法。其中 AbstractDb 存储用户历史浏览的摘要信息，存储的信息包括摘要数据 id，新闻 id，以及一个摘要转化成的字符串。

四．其他

NewsListAdapter 是一个 RecyclerView 适配器扩展，主要功能是展示新闻列表，包括绑定数据，加载图片，以及点击跳转新闻详情页的功能。

TabNewsFragment 主要用于实现顶部可以滑动类别的导航栏，包括实现上滑下滑加载动画，爬取数据，以及实现列表上下滑动的功能。

2 具体实现

一．　新闻主界面

Ui 通过一个 drawerlayout 实现侧滑抽屉效果，其中三个 item 分别跳转历史列表、收藏列表、设置类别界面，除抽屉外界面由 app_bar_main.xml 实现，其中包括一个 toolbar，跳转搜索界面的 buttontosearch，滑动类别导航栏通过 tab_layout 实现，下方新闻列表通过 viewPager 实现。

其中三个 item 和 buttontosearch 需要设置点击事件，以 buttontosearch 为例：

```
buttontosearch.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this, SearchActivity.class);
        startActivity(intent);
    }
});
```

二．　新闻列表

为了正确展示新闻列表，需要给主活动的 viewPager 设置一个 adapter。这里设置了一个字符串数组 titles 存储用户希望展示的全部新闻分类，其中 Fragment 应用 TabNewsFragment，其标题依次选取 titles 中每一项。并通过 tab_layout 点击事件实现动画效果。

```
//viewPager需要设置一个adapter
viewPager.setAdapter(new FragmentStateAdapter( fragmentActivity: this){
    @NonNull  1 usage
    @Override
    public Fragment createFragment(int position){
        String title=titles.get(position);
        TabNewsFragment tabNewsFragment=TabNewsFragment.newInstance(title);
        return tabNewsFragment;
    }
    @Override
    public int getItemCount() { return titles.size(); }
});
```

```
//tab_layout点击事件
tab_layout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener(){
    @Override  1 usage
    public void onTabSelected(TabLayout.Tab tab){
        //设置viewPager选中当前页
        viewPager.setCurrentItem(tab.getPosition(), smoothScroll: false);
    }
    @Override  1 usage
    public void onTabUnselected(TabLayout.Tab tab){

    }
    @Override  1 usage
    public void onTabReselected(TabLayout.Tab tab){

    }
});
```

再将 tab_layout 和 viewPager 关联起来，使得每个 tab 对应正确的新闻列表。

```
    //tab_layout和viewPager关联在一起
    TabLayoutMediator tablayoutMediator=new TabLayoutMediator(tab_layout, viewPager, new TabLayoutMediator.TabConfigurationStrategy() {
        @Override  1 usage
        public void onConfigureTab(@NonNull TabLayout.Tab tab, int position) {
            tab.setText(titles.get(position));
        }
    });
    tablayoutMediator.attach();
}
```

为了实现新闻列表的 ui，需要定义一个 news_list_item，使得列表中每条新闻有固定的呈现形式。

上下滚动的视图通过 RecyclerView 实现，刷新功能通过 RefreshLayout 实现。

首先继承一个 RecyclerView Adapter 类 NewsListAdapter。

通过 Myholder 快速加载布局文件

```
static class MyHolder extends RecyclerView.ViewHolder {  4 usages
    ImageView image;  2 usages
    TextView publisher;  2 usages
    TextView publishTime;  2 usages
    TextView title;  4 usages
    public MyHolder(@NonNull View itemView) {  1 usage
            super(itemView);
            image=itemView.findViewById(R.id.image);
            publisher=itemView.findViewById(R.id.publisher);
            publishTime=itemView.findViewById(R.id. publishTime);
            title=itemView.findViewById(R.id.title);
    }
}
```

```
@NonNull
@Override
public MyHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    //加载布局文件
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.news_list_item, root: null);
    return new MyHolder(view);
}
```

定义一个 NewsInfo 类动态数组 mDataDTOList，将希望在列表中展示的新闻全部通过 addListData 存入，进而绑定在新闻列表视图中。

```
public void addListData(List<NewsInfo.DataDTO> listData){  2 usages
    int startPosition=this.mDataDTOList.size();
    this.mDataDTOList.addAll(listData);
    notifyItemRangeChanged(startPosition, listData.size());
}
```

获取数据的方式通过 url 拼贴的形式，

baseUrl=https://api2.newsminer.net/svc/news/queryNewsList?size=10&startDate=2015-08-31&endDate=2024-09-02&words&categories=的基础上，我们需要新增正确的类别分类获取正确的数据，如果类别为“全部”则对 categories 值没有限制，否则需要 categories 与类别相同。通过创建 okhttp 对象和 request 对象获取数据。

```
private void getHttpData() throws UnsupportedEncodingException{  3 usages
    //创建OkHttpClient对象
    OkHttpClient okHttpClient = new OkHttpClient();
    String encodedtitle=Objects.equals(title,  b: "全部") ? "" : URLEncoder.encode(title, StandardCharsets.UTF_8.toString());
    String url=baseUrl+encodedtitle+"&page="+currentPage;
    //构造Request对象
    Request request = new Request.Builder()
            .url(url)
            .get()
            .build();
    //通过OkHttpClient和Request对象来构建Call对象
    Call call = okHttpClient.newCall(request);
    call.enqueue(new Callback() {
```

由于不能在耗时操作中更新 ui，所以我们需要通过 Handler 将获取的数据传递给 NewsListAdapter，进而在新闻列表中新添新闻。

```
@Override
public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
    String data=response.body().string();
    //不能在耗时操作里面更新ui，那么需要使用handler
    Message message=new Message();
    message.what=100;
    message.obj=data;
    //发送
    mHandler.sendMessage(message);
    isLoading=false;
}
```

此时我们将 mDataDTO 每条数据的的来源标题和第一张图片绑定并加载，如果点进该条新闻则通过点击事件将标题颜色变灰。特别地，这里 image_error 是一个我们已经预先设定好的图片，如果新闻自身没有图片则我们会在列表中加载这个图片，以保证美观。而在新闻列表中绑定和点击事件中两次设定标题颜色为灰则可以保证不管我们从哪个界面回到新闻列表，已经看过的新闻都会是灰色的。而在 holder 改变颜色时应该进行 else 判断，不在历史记录新闻应设置为黑色，防止所有新闻变为灰色。

```
@Override
public void onBindViewHolder(@NonNull MyHolder holder, @SuppressLint("RecyclerView") int position) {
    //绑定数据
    NewsInfo.DataDTO dataDTO= mDataDTOList.get(position);
    holder.publisher.setText("来源："+dataDTO.getPublisher());
    holder.publishTime.setText(dataDTO.getPublishTime());
    holder.title.setText("标题："+dataDTO.getTitle());
    if(HistoryDbHelper.getInstance(mContext).isHistory(dataDTO.getNewsID())){
        holder.title.setTextColor(Color.parseColor( colorString: "#A8A8A8" ));
    }else{
        holder.title.setTextColor(Color.BLACK);
    }
    //加载图片
    String img=dataDTO.getrealimage( i: 0);
    Glide.with(mContext).load(img).error(R.mipmap.image_error).into(holder.image);
    //点击事件
    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(null!=mOnItemClickListener){
                mOnItemClickListener.onItemClick(dataDTO,position);
                holder.title.setTextColor(Color.parseColor( colorString: "#A8A8A8" ));
            }
        }
    });
}
```

为了实现上拉下拉效果，我们定义一个 int 类型 currentPage，初始值设定为 1，每当需要加载新数据时我们增大 page 获取更多数据。

```
private void loadMoreItems() {  1 usage
    isLoading = true;
    currentPage++;
    try {
        getHttpData();
    } catch (UnsupportedEncodingException e) {
        Log.e( tag: "EncodingError",  msg: "Unsupported Encoding Exception", e);
    }
}
```

最后通过这种拼贴的 url 获取数据。

```
String encodedtitle=Objects.equals(title, b: "全部") ? "" : URLEncoder.encode(title, StandardCharsets.UTF_8.toString());
String url=baseUrl+encodedtitle+"&page="+currentPage;
```

设置滚动监视器，只有同时满足三个条件才认为滚动到了底部。防止了重复加载、

确保已经滚动到了底部。

```
//为RecyclerView设置滚动监视器
recyclerView.addOnScrollListener(onScrolled(recyclerView, dx, dy) → {
        super.onScrolled(recyclerView, dx, dy);
        int visibleItemCount = layoutManager.getChildCount();//获取了当前屏幕上可见的RecyclerView项的数量
        int totalItemCount = layoutManager.getItemCount();//获取了RecyclerView中所有项的总数。
        int firstVisibleItemPosition = layoutManager.findFirstVisibleItemPosition();//这行代码获取了当前屏幕上第一个可见项在适配器中的位置。

        if (!isLoading && (visibleItemCount + firstVisibleItemPosition) >= totalItemCount
                && firstVisibleItemPosition >= 0) {
            loadMoreItems();
        }////判断来检查是否滚动到了底部三个条件同时满足：1.确保当前没有正在加载数据的操作，避免重复加载
});//2.第一个可见项的位置加上屏幕上可见项的数量大于等于总项数时，可以认为已经滚动到底部3.确保第一个可见项的位置是有效的，即不是负值。
```

设置刷新监视器用于刷新，从顶部下拉则重新获取数据。

```
swipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    @Override  1 usage
    public void onRefresh() {
        currentPage = 1;
        try {
            getHttpData();
        } catch (UnsupportedEncodingException e) {
            Log.e( tag: "EncodingError",  msg: "Unsupported Encoding Exception", e);
        }
        swipeRefreshLayout.setRefreshing(false);
    }
});
```

三．列表分类

SettingActivity 界面设计了 12 个按钮，分别对应 12 种分类

每个按钮初始根据其在 PreferenceDb 是否存在设置颜色（有则证明上次用户不希望看到这个类别，置白色，否则为红）在点击时则修改相应词条在 PreferenceDb 中的状态（原来有则删，原来没有则添），并改变其颜色。这里以“科技”按钮为例。

```
//button1 科技
if(!PreferenceDbHelper.getInstance( context: SettingActivity.this).isPreference( preferencetype: "科技")){
    button1.setBackgroundColor(Color.RED);
}else{
    button1.setBackgroundColor(Color.WHITE);
}
button1.setOnTouchListener(new View.OnTouchListener() {
    @SuppressLint("ClickableViewAccessibility")
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            // 当按钮被按下时
            if (!PreferenceDbHelper.getInstance( context: SettingActivity.this).isPreference( preferencetype: "科技")) {
                button1.setBackgroundColor(Color.WHITE);
                PreferenceDbHelper.getInstance( context: SettingActivity.this).addPreference( preferencetype: "科技");
            } else {
                button1.setBackgroundColor(Color.RED);
                PreferenceDbHelper.getInstance( context: SettingActivity.this).deletepreference( preferencetype: "科技");
            }
        }
        return false;
    }
});
```

之后为确定按钮设计点击事件，单击后颜色变红并且立刻跳转回主界面。特别

地，为了保证此次修改可以立刻更改主界面看到的新闻类别而不需要重启应用，这里通过 intent 重启了主界面。

```
findViewById(R.id.buttonsure).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        buttonsure.setBackgroundColor(Color.RED);
        Intent intent = new Intent( packageContext: SettingActivity.this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
    }
});
```

相应地，在主活动中，titles 应当每次都是一个空的字符串动态数组，只有用户希望看到某个类别时，才将其加入 titles 展示。

```
List<String> titles=new ArrayList<>();
if(!PreferenceDbHelper.getInstance( context: MainActivity.this).isPreference( preferencetype: "科技")){
    titles.add(new String( original: "科技"));
}
```

三. 新闻详情

布局文件 activity_news_details 包括一个上方的星型按钮用于添加收藏，五个 textView 分别用于显示标题、来源、时间、新闻文字内容以及摘要。

```
// recyclerView列表点击事件
mNewsListAdapter.setOnItemClickListener(new NewsListAdapter.onItemClickListener() {
    @Override  1 usage
    public void onItemClick(NewsInfo.DataDTO dataDTO, int position) {
        //跳转到详情页
        Intent intent = new Intent(getActivity(), NewsDetailsActivity.class);
        //传递对象的时候，该类一定要实现serializable
        intent.putExtra( name: "dataDTO",dataDTO);
        startActivity(intent);


    }
});
```

相关数据已经在 TabNewsFragment 中的点击事件通过 intent 方法传递过来了，这里只需要接收并加载。

```
//获取传递的数据
dataDTO = (NewsInfo.DataDTO) getIntent().getSerializableExtra( name: "dataDTO");
//设置数据
if (null != dataDTO) {
    toolbar.setTitle(dataDTO.getTitle());
    textViewtitle.setText(dataDTO.getTitle());
    textViewsource.setText("来源：" + dataDTO.getPublisher());
    textViewtime.setText("时间：" + dataDTO.getPublishTime());
    textViewcontent.setText(dataDTO.getContent());
```

由于新闻中可能有多张图片，我们需要使用动态添加图片的方式添加图片，这样才能保证所有图片不会被最后一张图片覆盖。

```
// 动态添加ImageView并加载图片
for (String url : imageUrls) {
    ImageView imageView = new ImageView( context: this);
    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT);
    imageView.setLayoutParams(params);
    imageView.setAdjustViewBounds(true); // 根据图片的实际大小调整ImageView的大小（可选）
    imageView.setScaleType(ImageView.ScaleType.CENTER_CROP); // 图片的缩放类型（可选）
    // 使用Glide加载图片
    Glide.with( activity: this).load(url).into(imageView);
    // 将ImageView添加到LinearLayout中
    layoutContainer.addView(imageView);
}
```

这里需要特别说明的是，由于 api 所给的图片 url 以一个奇怪的列表字符串形式呈现，为了成功加载图片，我们需要先将其转化成一个正确的字符串列表，其中含有的都是正确的、可以加载的 URL。在 NewsInfo 中 我们自行定义了两个函数用于获取图片数量和第 i 张图片的 url。

```
public int getImagenum() {  2 usages
    String[] arr1 =image.split( regex: ",");
    //System.out.println(arr1.length);
    return arr1.length;
}
```

```
public String getrealimage(int i){  2 usages
    int num=getImagenum();
    if(num==0){
        return "";
    } else{
        int len=image.length();
        String tmp=image.substring(1,len-1);
        String arr2[]=tmp.split( regex: ",");
        if(i!=0){
            arr2[i]=arr2[i].substring( beginIndex: 1);
        }
        return arr2[i];
    }
}
```

只需要把所有 url 添加入空字符串列表即可。

```
ArrayList<String> imageUrls = new ArrayList<>();
int num = dataDTO.getImagenum();
for (int i = 0; i < num; i++) {
    imageUrls.add(dataDTO.getrealimage(i));
}
```

之后我们再来动态添加视频，需要说明的是，由于部分视频过小，我们需要人为设定这个图片的宽度和高度。

```
//播放视频
if(dataDTO.getVideo()!=""){
    VideoView videoView1 = new VideoView( context: this);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);    //全屏
    // 获取当前屏幕的密度
    float density = getResources().getDisplayMetrics().density;

    // 将 dp 转换为 px
    int widthInPx = (int) (450 * density + 0.5f);
    int heightInPx = (int) (300 * density + 0.5f);

    // 创建 LinearLayout.LayoutParams 对象
    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
            widthInPx, // 使用转换后的像素值
            heightInPx); // 使用转换后的像素值

    videoView1.setLayoutParams(params);
    Uri uri = Uri.parse(  dataDTO.getVideo());
    videoView1.setVideoURI(uri);
    videoView1.requestFocus();
    videoView1.start();
    layoutContainer.addView(videoView1);
}
```

在使用 api 生成模型摘要时，我们同样通过创建 okhttp 对象实现。请求头格式根据质谱 ai 官网格式生成。需要特别说明的是，其中的 text 来自新闻中的 content，但必须去除其中的换行符\n，否则将不符合模型输入格式，生成摘要失败。

```
private void sendPostRequest(String text,String newsID) {  1 usage
    text="你好,请根据以下内容为我生成一篇不多于300字的摘要"+text;
    //需要将新闻文本中所有换行符\n去除 否则将无法生成摘要
    text = text.replaceAll( regex: "\\r?\\n",  replacement: "");
    OkHttpClient client = new OkHttpClient();

    MediaType JSON = MediaType.parse( $this$parse: "application/json; charset=utf-8");
    String jsonData = String.format("{\"model\": \"glm-4\",\"messages\": [{\"role\": \"user\",\"content\": \"%s\"}]}", text);
    System.out.println(jsonData);
    RequestBody body = RequestBody.create(jsonData, JSON);

    Request request = new Request.Builder()
            .url("https://open.bigmodel.cn/api/paas/v4/chat/completions")
            .header( name: "Authorization",  value: "Bearer d9e8bcee0e8723b89c16c7bd2ef08199.wRiKecdRi6i3ttis") // 替换为你的实际to
            .post(body)
            .build();
```

```java
@Override
public void onResponse(Call call, Response response) throws IOException {
    if (!response.isSuccessful()) {
        runOnUiThread(() -> textViewabstract.setText("请求未成功: " + response.code()));
    } else {
        // 假设响应是一个简单的JSON字符串
        String responseBody = response.body().string();
        System.out.println(responseBody);
        // 创建ObjectMapper实例
        ObjectMapper objectMapper = new ObjectMapper();

        // 读取JSON字符串为JsonNode对象
        JsonNode rootNode = objectMapper.readTree(responseBody);

        // 导航到choices数组的第一个元素
        JsonNode choicesNode = rootNode.path( fieldName: "choices").get(0);

        // 从choices的第一个元素中提取message对象
        JsonNode messageNode = choicesNode.path( fieldName: "message");

        // 从message对象中提取content字段
        content = messageNode.path( fieldName: "content").asText();
        //JSONObject jsonObj=Json.parseObject(responseBody);
        // 在这里处理响应数据，然后更新UI
        // 假设这是你的responseBody，一个包含content字段的JSON字符串
        runOnUiThread(() -> textViewabstract.setText("新闻摘要（由模型glm-4生成）: " + content));
        AbstractDbHelper.getInstance( context: NewsDetailsActivity.this).addAbstract(newsID,content);
    }
```

需要特别说明的是，有时候新闻内容可能较长，导致模型响应超时。这里我们需要在 sendPostRequest 里修改响应时间以防超时。

```java
OkHttpClient client = new OkHttpClient();
client = new OkHttpClient.Builder()
        .readTimeout( timeout: 20, TimeUnit.SECONDS)
        .connectTimeout( timeout: 20, TimeUnit.SECONDS)
        .build();
```

在读取结果时，我们只读取模型相应的内容中的 choices 的 message 对象的 content 字段，得到摘要，并用其更新 ui。这里必须说明的是，网络请求部分必须单独在 oncreate 外写一个函数，不能直接用在 ui 更改界面，否则将导致空指针错误。在 textView 修改时只能通过调用 sendPostRequest 获得生成的摘要。

为了实现本地存储生成的摘要，我们新建一个数据库 AbstractDb，存储新闻 id 和生成的摘要。模型生成摘要后将其添加入数据库，AbstractDb 需要 SQLite 的增删改查方法。这里只展示有用的增和查方法。

```java
/**
 * 判断是否生成过摘要
 */
@SuppressLint("Range")  2 usages
public boolean isAbstract(String newsID) {
    //获取SQLiteDatabase实例
    SQLiteDatabase db = getReadableDatabase();
    String sql = "select Abstract_id,newsID,Abstract  from abstract_table where newsID=?";
    String[] selectionArgs = {newsID};
    Cursor cursor = db.rawQuery(sql, selectionArgs);
    //cursor.close();
    //db.close();
    return cursor.moveToNext();
}
```

```java
@SuppressLint("Range")  2 usages
public String getAbstractByNewsID(String newsID) {
    SQLiteDatabase db = this.getReadableDatabase();
    String query = "SELECT Abstract FROM abstract_table WHERE newsID = ?";

    String[] args = {newsID};
    Cursor cursor = db.rawQuery(query, args);

    String abstractText = null;
    if (cursor.moveToFirst()) {
        abstractText = cursor.getString(cursor.getColumnIndex( columnName: "Abstract"));
    }
    cursor.close();
    db.close();

    return abstractText;
}
```

为了保证不反复请求 api，我们在新闻详情界面生成摘要时需要进行判断。如果摘要数据库里没有对应的 newsID，则需要生成摘要。此外，有可能存在用户第一次浏览时网络等因素造成生成摘要失败的情况，这种情况数据库中摘要为 null，也需要重新生成。

```
//生成ai摘要
String text=dataDTO.getContent();
String newsID=dataDTO.getNewsID();
if(AbstractDbHelper.getInstance( context: NewsDetailsActivity.this).isAbstract(dataDTO.getNewsID())==false||AbstractDbHelper.getInst
    sendPostRequest(text,dataDTO.getNewsID());
} else{
    String abstractText =AbstractDbHelper.getInstance( context: NewsDetailsActivity.this).getAbstractByNewsID(dataDTO.getNewsID());

    textViewabstract.setText("摘要已生成:"+abstractText);
}
```

通过这种方法，我们所有生成的摘要也完全存储到了数据库中，断网依然留存。

四． 本地记录

首先是本地历史记录，我们在主界面同样通过 SQLite 的增添方法将新闻信息添加至 HistoryDb 中。

```
//添加历史记录
String dataDTOJson = new Gson().toJson(dataDTO);
HistoryDbHelper.getInstance( context: NewsDetailsActivity.this).addHistory( username: null, dataDTO.getNewsID(), dataDTOJson);
```

这里 HistoryInfo 存储的信息包括 newsID 和将整个新闻数据转化得的 Json 类型 dataDTOJson。SQLite 增删改查方法与前面的摘要数据库类似，这里不再赘述。HistoryListActivity 的界面则和 TabNewsFragment 类似，也要使用 NewsListAdapter，不过获取数据的方式则是从数据库中调取。

```
//获取数据
List<HistoryInfo> historyInfoList=HistoryDbHelper.getInstance( context: HistoryListActivity.this).queryHistoryListData( username: null);
Gson gson=new Gson();
for(int i=0;i< historyInfoList.size();i++){
    if(historyInfoList.get(i).getNew_json()!=null)mDataDTOList.add(gson.fromJson(historyInfoList.get(i).getNew_json(),NewsInfo.DataDTO.class));
}
```

之后是收藏列表，在新闻详情界面中，我们通过左上角星星按钮的点击事件来更改收藏数据库 CollectionDb。原理和 SettingActivity 中按钮相同。每次点击变色同时修改其在数据库中状态，初始显示状态根据用户上次设定决定。

```
//添加收藏
if (!CollectionDbHelper.getInstance( context: NewsDetailsActivity.this).isCollection(dataDTO.getNewsID())) {
    myButton.setBackgroundResource(R.drawable.button_notselected);
} else {
    myButton.setBackgroundResource(R.drawable.button_selected);
}
myButton.setOnTouchListener(new View.OnTouchListener() {
    @SuppressLint("ClickableViewAccessibility")
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            // 当按钮被按下时
            if (!CollectionDbHelper.getInstance( context: NewsDetailsActivity.this).isCollection(dataDTO.getNewsID())) {
                myButton.setBackgroundResource(R.drawable.button_selected);
                isSelected = true;
                CollectionDbHelper.getInstance( context: NewsDetailsActivity.this).addCollection( username: null, dataDTO.getNewsID(), dataDTOJson);
            } else {
                myButton.setBackgroundResource(R.drawable.button_notselected);
                CollectionDbHelper.getInstance( context: NewsDetailsActivity.this).delete(dataDTO.getNewsID());
                isSelected = false;
            }
        }
        return false;
    }
});
```

CollectionDbHelper 也与前面的 SQLite 修改方法类似，不过需要一个额外的删除收藏记录的方法。

```
/**
 * 根据newsID删除收藏记录
 */
public int delete(String newsID) {  1 usage
    //获取SQLiteDatabase实例
    SQLiteDatabase db = getWritableDatabase();
    // 执行SQL
    int delete = db.delete( table: "collection_table",  whereClause: " newsID=?", new String[]{newsID});
    // 关闭数据库连接
    db.close();
    return delete;
}
```

需要特别说明的是，在 CollectionListActivity 中，我们可能会通过收藏列表点进一个新闻详情页，然后在该新闻详情页更改这则新闻的收藏状态，此时如果像之前的历史列表活动一样把所有代码全部写在 oncreate 方法中可能导致从新闻详情退回后新闻列表更新不及时（因为 ui 不能及时根据数据库的变化更新），为了实现这种情况下新闻列表的及时更新，我们需要使用 onresume 方法。

```
protected void onResume() {
    super.onResume();
    //获取数据
    List<CollectionInfo> collectionInfoList= CollectionDbHelper.getInstance( context: CollectionListActivity.this).queryCollectionListData( username: null);
    mDataDTOList.clear(); // 清除旧数据
    Gson gson=new Gson();
    for(int i=0;i< collectionInfoList.size();i++){
        mDataDTOList.add(gson.fromJson(collectionInfoList.get(i).getNew_json(),NewsInfo.DataDTO.class));
    }
    //设置数据
    mNewsListAdapter.setListData(mDataDTOList);
}
```

可以看到，我们把获取数据和设置数据的部分全部放进了 onresume，注意此处要及时清除旧数据，否则可能出现一则新闻在收藏列表中多次出现的问题。

五．新闻搜索

SearchActivity 界面包括关键字输入框、开始日期选择按钮、结束日期选择按钮、类别输入框。

其中关键词搜索框和搜索类别输入框可以通过这种方式实现，在点击输入时提示消失。

```
// 关键词
search_keyword.setOnFocusChangeListener((v, hasFocus) -> {
    if (hasFocus) {
        search_keyword.setHint("");
    } else {
        search_keyword.setHint("请输入关键词");
    }
});
```

日期选择按钮则可以通过这种方式实现在日历上选择。

```
// 初始化日期按钮文本
updateDateButtonText();
```

```
// 设置日期按钮
start_date_button.setOnClickListener(v -> showDatePickerDialog( isStart: true));
end_date_button.setOnClickListener(v -> showDatePickerDialog( isStart: false));
```

再为搜索按钮设置点击事件，并跳转搜索结果界面，通过 intent 在两个活动之间传递搜索条件信息。

```
search_button.setOnClickListener(v -> {
    String keyword = search_keyword.getText().toString().trim();
    String startDate = start_date_button.getText().toString().replace("start_date" + ": ", replacement: "").trim();
    String endDate = end_date_button.getText().toString().replace("end_date" + ": ", replacement: "").trim();
    String category = search_category.getText().toString().trim();


    Toast.makeText( context: this, text: "搜索关键词: " + keyword + ", 开始日期: " + startDate + ", 结束日期: " + endDate + ", 类别: " + category, Toast.LENGTH_LONG).show();
    Intent intent = new Intent( packageContext: SearchActivity.this, SearchResultActivity.class);
    intent.putExtra( name: "KEYWORD", keyword);
    intent.putExtra( name: "START_DATE", startDate);
    intent.putExtra( name: "END_DATE", endDate);
    intent.putExtra( name: "CATEGORY", category);
    startActivity(intent);
});
```

SearchResultActivity 同样类似 TabNewsFragment，使用 NewsListAdapter 展示新闻列表，不过数据的传递是通过 indent 接收来自 SearchActivity 的数据，然后根据这些搜索条件重新拼贴 url，重新进行新闻数据的爬取。

```
Intent intent = getIntent();
keyword = intent.getStringExtra( name: "KEYWORD");
cat = intent.getStringExtra( name: "CATEGORY");
startDate = intent.getStringExtra( name: "START_DATE");
endDate = intent.getStringExtra( name: "END_DATE");
```

```
private void getHttpData() throws UnsupportedEncodingException{ 3 usages
    //创建OkHttpClient对象
    OkHttpClient okHttpClient = new OkHttpClient();
    //构造Request对象
    String url=baseUrl+"&startDate="+startDate+"&endDate="+endDate+"&words="+keyword+"&categories="+cat+"&page="+currentPage;
    Request request = new Request.Builder()
            .url(url)
            .get()
            .build();
```

3 总结与心得

在小学期期间，我主要学习了 java 相关语法和安卓开发相关知识，并从零完成了一个大作业——安卓新闻客户端的开发。在这一过程中，我主要有以下收获：首先，我学会了如何将课上所学的知识熟练运用在实际开发中，字符串、动态数组等相关语法在我的整体代码中相当重要。其次，我做到了通过多种方式获取项目所需要的、课堂未能展开的知识，以此实现项目的多种功能。比如我在 csdn 获取了很多 ui 设计相关的知识，通过询问大模型获取了动态添加图片、调用 glm 的方法，通过在群里和同学们讨论发现了很多之前没有注意到的 bug（比如图片的 URL 格式等问题）。最后，我也锻炼了独立摸索、探索项目的能力，每当运行遇到结果不尽如人意时，我通过查看 logcat 报错、查看数据库、以及注释、输出等多种方式调试以期获知问题所在，这一过程也增长了我独立调试代码的能力。

关于 project 我主要有以下建议：首先，今年 glm 生成摘要这个环节非常好，与当下最新技术发展相结合，我个人认为是一次非常有意义的创新，希望明年可以沿用。其次，项目本身在实现过程中存在一些自身的问题，一定程度上耗费了我一些不必要的时间，比如 url 获取的新闻图片格式、视频链接自身无法播放等等，我希望明年可以提供一个更新更完善的 api 给下届同学们，完善大家完成大作业的体验。