中南大學
CENTRAL SOUTH UNIVERSITY

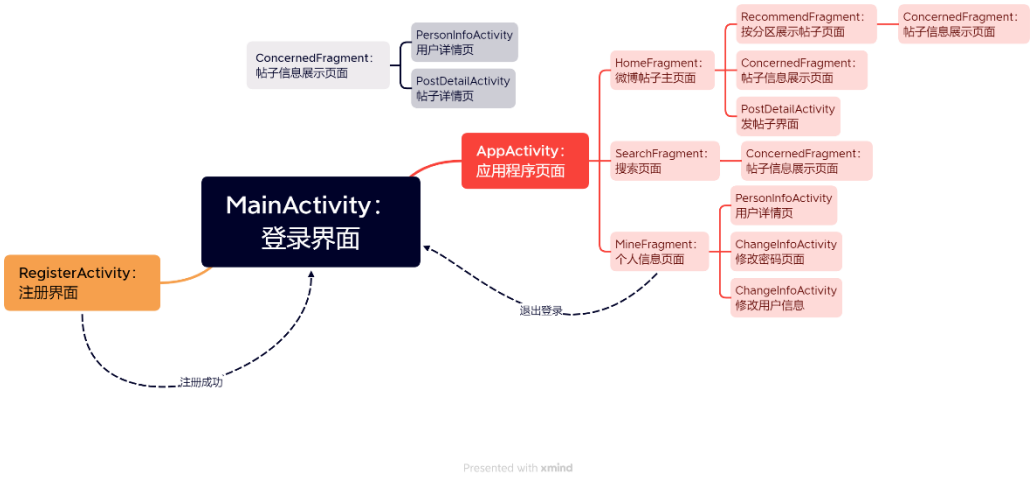
计算机应用实践
课程设计实验报告

学生姓名	
学　　号	
专业班级	
指导教师	
学　　院	
完成时间	

计算机学院

目录

一、微博应用业务分析（OK）



本项目主要完成微博的用户功能和帖子功能两大模块。项目中所有的输入部分都需要有效性检验，后续不再赘述。

首先是登录注册功能，登录时用户发送账号密码提交服务器验证返回结果，成功则跳转进 app 主界面。注册时要注意后端需要验证该账号是否已经被注册了，注册界面用一个表单收集用户基本信息然后上传服务器保存，接着跳转登录界面。

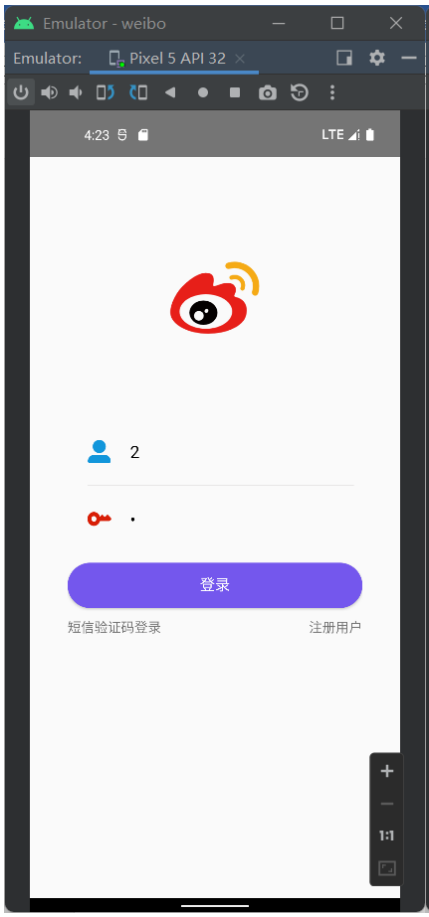
在主界面中用 tab 布局分三个模块：展示帖子，查询帖子，个人信息。

第一个界面可以点击右上方的加号进行发帖功能，帖子支持文字和图片，图片以九宫格的形式展示。同时用 tab 布局分两个模块：关注模块展示关注用户的发的帖子；推荐模块按照分区展示帖子。在帖子展示界面点击头像可以进入用户个人空间，点击帖子可以进入帖子详情页，用户可以对一条帖子进行评论，该项目设计上支持两级评论功能。

第二个界面有一个搜索框，根据用户输入的信息进行帖子查询，展示包含查询内容的所有帖子。

第三个界面展示用户个人信息，在此界面处可以跳转修改账号密码界面，修改个人信息界面，用户个人空间界面。同时可以在这里进行账号的退出。在用户个人空间中用 tab 页展示个人详细信息和用户发的所有帖子，同时点击用户头像可以进行更换头像的操作。

二、微博应用运行结果与测试分析（OK）


Emulator - weibo
Emulator: Pixel 5 API 32
4:23
LTE
2
登录
短信验证码登录
注册用户

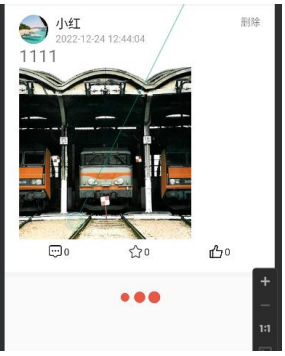

Emulator - weibo
Emulator: Pixel 5 API 32
4:24
LTE
昵称 不超过10个字符
性别 男 女
电话
介绍 请介绍一下自己
账号 请输入账号
密码 请输入密码
注册


Emulator - weibo
Emulator: Pixel 5 API 32
4:24
LTE
关注
推荐
小红
2022-12-24 16:07:15
第三条微博
+6
0
0
0
小红
2022-12-24 16:06:38
222
首页
发现
我的


Emulator - weibo
Emulator: Pixel 5 API 32
4:25
LTE
关注
推荐
热门
世界杯
同城
榜单
明星
小明
2022-12-21 19:01:55
这是一条世界杯的微博
0
0
0
首页
发现
我的


Emulator - weibo
Emulator: Pixel 5 API 32
4:41
LTE
取消
发布帖子
发表
发送一条微博帖子11111


4:29
LTE
关注
推荐
小红
2022-12-26 11:40:16
测试下评论

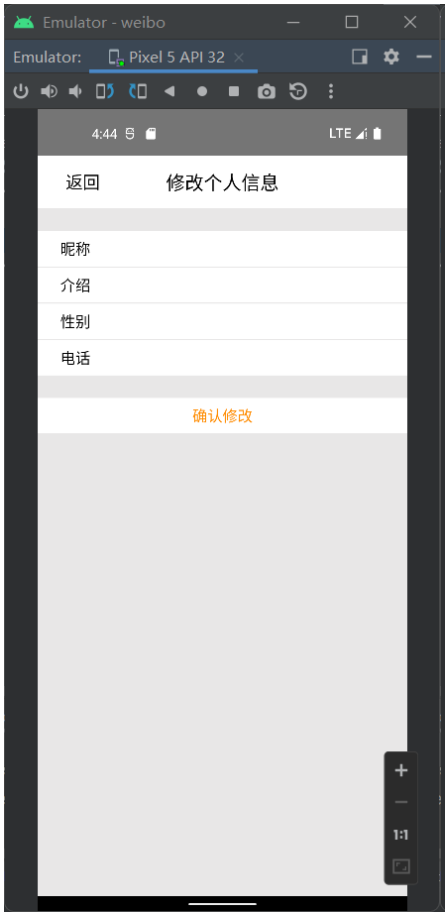

小红
2022-12-24 12:44:04
删除
1111
0
0
0


Emulator - weibo
Emulator: Pixel 5 API 32
4:26
LTE
111
小红
2022-12-24 12:44:04
1111
0
0
0
小明
2022-12-03 19:21:06
首页
发现
我的

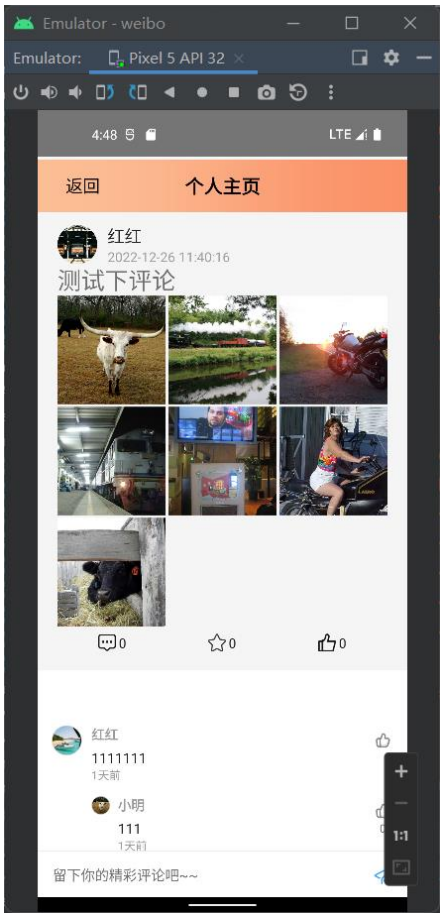

Emulator - weibo
Emulator: Pixel 5 API 32
4:26
LTE
小红
我是2号用户
0
微博
0
关注
0
粉丝
修改密码
修改个人信息
退出登录
首页
发现
我的

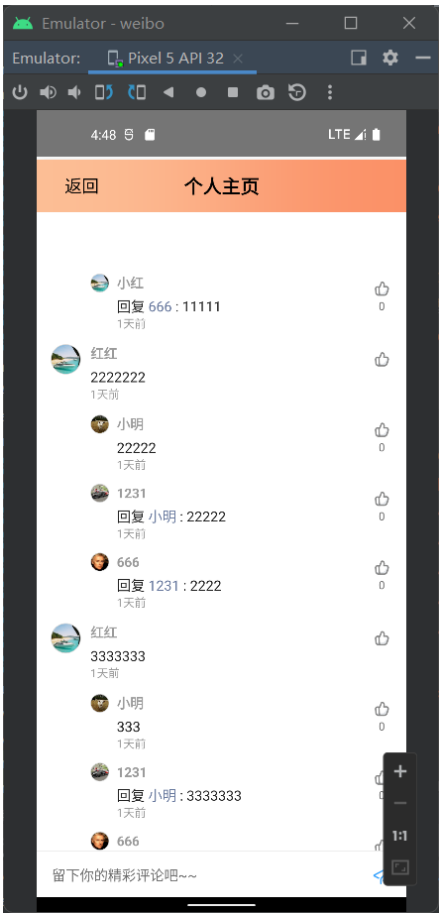

Emulator - weibo
Emulator: Pixel 5 API 32
4:27
LTE
返回
个人主页
小红
关注：0 | 粉丝：0
简介：无
个人信息
微博
昵称 小红
介绍 我是2号用户
性别 女
电话 135135153


Emulator - weibo
Emulator: Pixel 5 API 32
4:27
LTE
返回
个人主页
小红
关注：0 | 粉丝：0
简介：无
个人信息
微博
删除
小红
2022-12-26 11:40:16
测试下评论


Emulator - weibo
Emulator: Pixel 5 API 32
4:44
LTE
返回
修改个人信息
昵称
介绍
性别
电话
确认修改

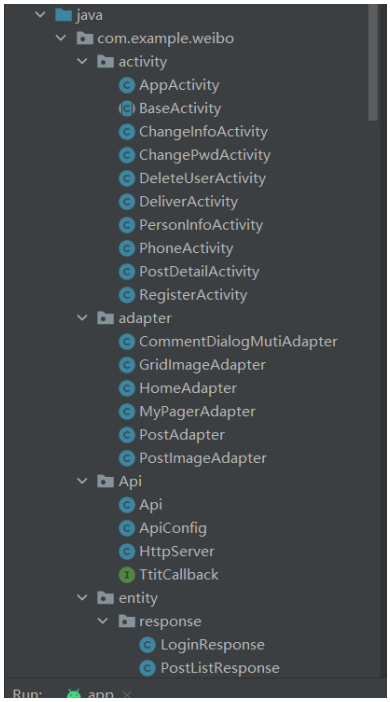

Emulator - weibo
Emulator: Pixel 5 API 32
4:44
LTE
返回
修改密码
旧密码
新密码
确认新密码
确认修改

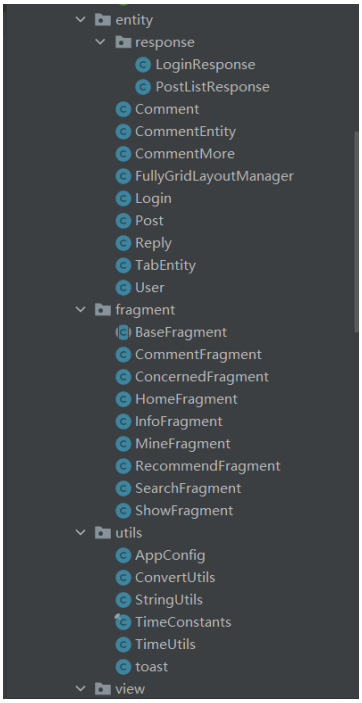


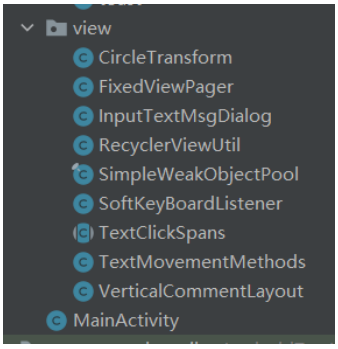


三、微博应用源代码清单（OK）

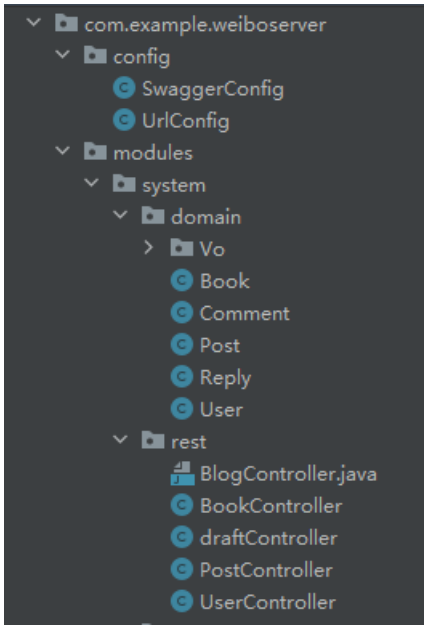
前端：

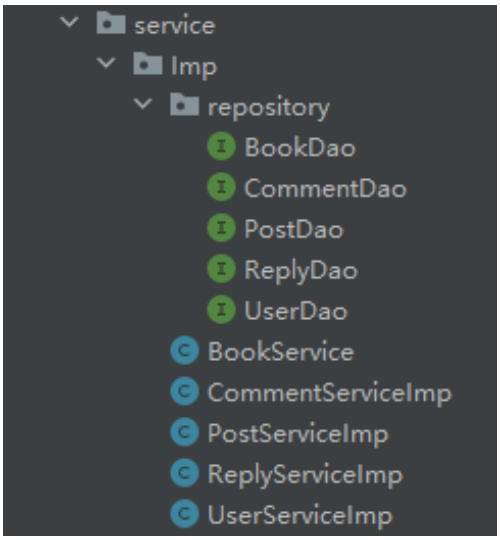






后端：





四、微博应用后端设计（OK）

数据库设计（OK）

User 用户表字段如下：

```
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;        //id
    private String profileUrl;      //头像图片的 Url，公共路径相同，存文件名即可
    private String introduction;      //介绍
    private String name;      //姓名
    @Column(unique=true)
    private String account;        //账号
    private String password;      //密码
    private String sex;        //性别
    private String phone;     //电话
    private Integer post_count;        //点赞数
    private Integer concerned_count;      //关注数
    private Integer fans_count;        //粉丝数
}
```

Post 帖子表字段如下：

```
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Integer id;                    //帖子的 id
    private Integer uid;                   //用户的 id
    private String profileURL;             //头像的 url
    private String name;                   //用户名
    private String time;                   //时间
    @Column(columnDefinition = "text")
    private String text;                   //正文
    @Column(columnDefinition = "text")
    private String pics;                   //图片链接集合，公共路径相同，存文件名即可
    private Integer likeCount;             //点赞数
    private Integer collectCount;          //收藏数
    private Integer commentCount;          //评论数
}
```

Comment 评论表字段如下：

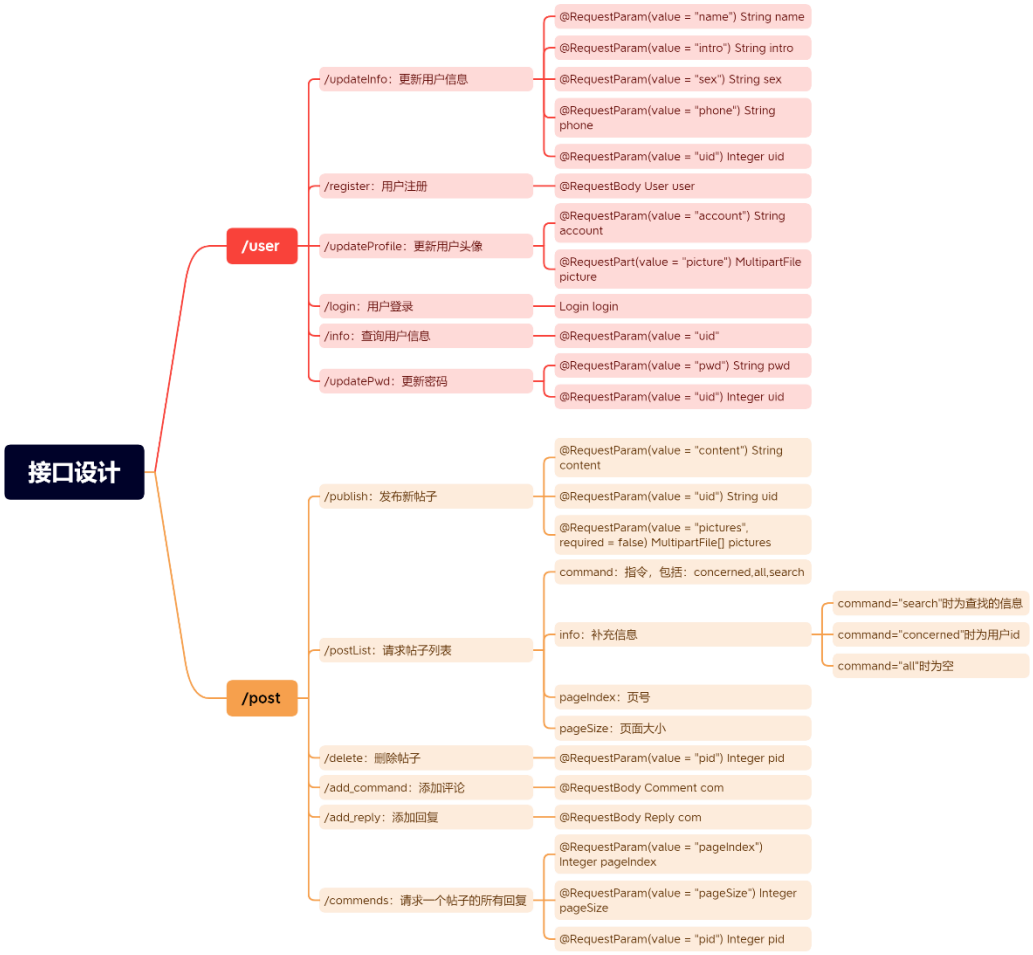
```
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;                    //父评论的 id
    private Integer pid;                   //评论帖子的 id
    private Integer uid;                   //评论用户的 id
    private String profileURL;             //评论者头像的 url
    private String name;                   //评论者用户名
    @Column(columnDefinition = "text")
    private String text;                   //评论的内容正文
    private Integer likeCount;             //评论的点赞数
    private Long time;                     //评论时间
}
```

Reply 回复表字段如下：

```
public class Reply {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;                    //评论的 id
    private Integer fid;                   //父评论的 id
    private Integer uid;                   //评论用户的 id
    private String profileURL;             //评论者头像的 url
    private String name;                   //评论者用户名
    private String fname;                  //父评论者用户名
    @Column(columnDefinition = "text")
    private String text;                   //评论的内容正文
    private Integer likeCount;             //评论的点赞数
    private Integer isReply;               //本条评论是否为回复
    private Long time;                     //评论时间
}
```

后端接口设计（OK）

后端接口及其需要的参数设计如下：



Presented with xmind

其中大部分功能都是比较简单的增删改查处理，需要注意的就是对修改操作要加@Transactional 和@Modifying 注解；登录时查询数据库看用户是否存在，再对比密码正确与否；注册时要根据账号检验用户是否已存在。此处仅分析较为复杂的三个/postList 接口，/commends 接口和/publish 接口。

/postList 接口请求微博帖子数据，根据业务情况帖子数据有四种应用：查询所有帖子；查询关注对象的帖子；查询某个用户的帖子；查询包含某些内容的帖子。由此设计 command 字段，不同应用携带的额外信息不同，额外信息存入 Info 参数中。根据对应的应用设计增删改查代码，这里使用 JPA 框架大部分都不需要书写 SQL 语句，需要注意的是查询包含某些内容的帖子使用到模糊查询：

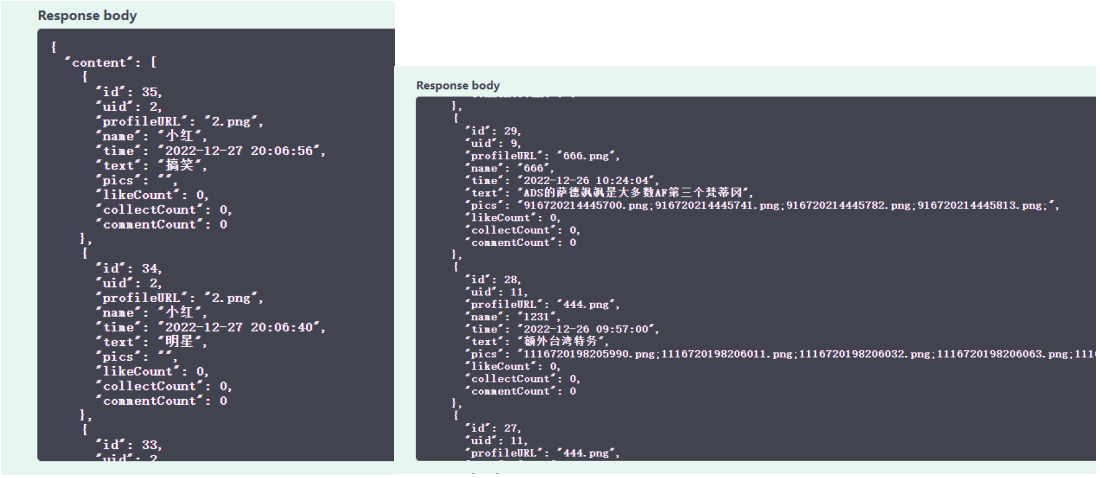
Page<Post> findByTextLikeOrNameLike(Pageable pageable, String info1, String info2);

这里所查询的 info 前后要加“%”。该接口的 controller 部分代码如下：

```
@ApiOperation("请求帖子列表")
@PostMapping("/postList")
```

```
public ResponseEntity<Object> postList(@RequestParam(value = "pageIndex") String pageIndex,
                                       @RequestParam(value = "command") String command,
                                       @RequestParam(value = "info") String info,
                                       @RequestParam(value = "pageSize") String pageSize){
    System.out.println(command+":"+info);
    Page<Post> postPage;
    if(command.equals("all")){        //查询所有帖子
        postPage = postService.findAllOrderTime(Integer.valueOf(pageIndex),Integer.valueOf(pageSize));
    } else if (command.equals("concerned")) {          //查询关注博主的帖子
        postPage = postService.findAllOrderTime(Integer.valueOf(pageIndex),Integer.valueOf(pageSize));
    } else if (command.equals("user")) {              //查询某个用户的所有帖子
        postPage = postService.findAllByUser(Integer.valueOf(pageIndex),Integer.valueOf(pageSize),Integer.valueOf(info));
    }
    else {          //查询搜索的帖子
        postPage = postService.findAllSearch(Integer.valueOf(pageIndex),Integer.valueOf(pageSize),"%"+info+"%");
    }
    for(Post post:postPage){
        post.setName(userService.findById(post.getUid()).getName());
    }
    return new ResponseEntity<>(postPage, HttpStatus.OK);
}
```

该接口返回数据形如：



/commends 接口返回评论数据。一篇微博下面可能会有多条评论。每条评论其实分为两种，一种是直接对博客的评论，称之为**父评论**，这里称为评论 Comment。另一种是对已有评论的评论，称为**子评论**，这里称为回复 Reply。

一篇微博可以有多条评论，一条父评论，可以有多条子评论。所以博客与评论的关系为一对多，父评论与子评论的关系也是一对多。评论的功能较为复杂。在这里数据库设计两张表，一张父表 Comment，一张子表 Reply。Comment 存储父评论，包括评论帖子的 id 和父评论的 id；Reply 存储子评论，包括父评论 id 和子评论 id。

请求评论时，后端接口接收三个参数 pid（帖子的 id），页号，页大小。先用 pid 在父表中搜索该帖子对应页内的父评论集 set1，遍历 set1 根据 uid 查询评论者的 name，然后取出 set1 中的 id 在子表中查询相关的所有子评论 set2。遍历 set2，取出 set2 的每个子元素，根据 fid（父评论 id）将其分配到对应父评论对象的列表中。最终接口返回列表数据，每个元素是一个父评论，每个父评论中有一个子评论对象列表。接口返回数据形式如下：

/publish 接口完成发布新帖子功能，需要注意的就是前端发送的图片文件列表要使用 MultipartFile[]类型来接收，遍历每个图片文件，使用 img.transferTo(saveFile)保存到资源文件夹下，saveFile 为每个文件的存储路径，文件名命名为时间戳+用户 id+第几张图片，以确保图片不重复。然后新建一个 post 对象持久化存储到数据库中：

```
@ApiOperation("发布新帖子")
@PostMapping("/publish")
public ResponseEntity<Object> update(@RequestParam(value = "content") String content,
                                     @RequestParam(value = "uid") String uid,
                                     @RequestParam(value = "pictures", required = false) MultipartFile[] pictures) {
    String picsUrl = "";        //所有图片的保存路径
    boolean uploadSuccess = true;
    String time = sdf.format(new Timestamp(System.currentTimeMillis()));      //获取当前时间
    if (pictures != null) {
        //存储每张图片的名字
        List<String> imageNames = new ArrayList<>();
        StringBuilder sb = new StringBuilder();
        for (int i =0;i<pictures.length;i++) {
```

```
                MultipartFile img = pictures[i];
                Long timeStamp = System.currentTimeMillis();
                String  saveUri=uploadPath+uid+timeStamp+i+".png";              //拼接保存图
片的真实地址:uid+时间戳+序号+".png"
                imageNames.add(uid+timeStamp+i+".png");
                log.info("保存一张图片到地址：  "+saveUri);
                File saveFile = new File(saveUri);
                try {
                    img.transferTo(saveFile);   //保存文件到真实存储路径下
                } catch (IOException e) {
                    uploadSuccess = false;
                    e.printStackTrace();
                }
            }
            for (String name : imageNames) {
                sb.append(name).append(";");
            }
            picsUrl = sb.toString();      //所有图片的保存路径
        }
        if(uploadSuccess){
            Post post = new Post();
            Integer id = Integer.valueOf(uid);
            User user = userService.findById(id);
            //新建帖子对象
            post.setName(user.getName());
            post.setUid(user.getId());
            post.setProfileURL(user.getProfileUrl());
            post.setPics(picsUrl);
            post.setText(content);
            post.setTime(time);
            post.setCollectCount(0);
            post.setCommentCount(0);
            post.setLikeCount(0);
            postService.addPost(post);
            return new ResponseEntity<>("OK", HttpStatus.OK);
        }
        else{
            return new ResponseEntity<>("NO",HttpStatus.NO_CONTENT);
        }
}
```

资源文件配置（OK）

在 D 盘中新建一个资源文件夹 weibo_resources，将资源文件配置到 weibo_resources 中，里面建两个文件夹，post_img 存放帖子的图片，profile 存用户头像。其中帖子图片命名格式为用户 id+时间戳+第几张图片+.png；用户头像图片命名为用户账号+.png。注意到由于资源存储路径是公共的，所以数据库中只存储文件名即可，这样**便于项目迁移时变更资源目录**。UrlConfig 类代码：

```
public class UrlConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        //addResourceHandler 是指你想在 url 请求的路径
        //addResourceLocations 是图片存放的真实路径

registry.addResourceHandler("/post_img/**").addResourceLocations("file:/D:/weibo_resources/post_img/");

registry.addResourceHandler("/profile/**").addResourceLocations("file:/D:/weibo_resources/profile/");
    }
}
```

Yml 文件配置：

```
web:
  upload-postimg: D:/weibo_resources/post_img/     # 存储博客的图片
  upload-profile: D:/weibo_resources/profile/   # 存储用户头像的图片
```

五、微博应用前端设计

BaseActivity 和 BaseFragment 的封装（OK）

考虑到所有的 activity 和 fragment 都有一些公共的操作，为了减少代码冗余，将这些通用操作抽取出来封装出公共基类 BaseActivity 和 BaseFragment。封装公共操作有：

- 初始化操作：初始化布局，绑定控件，绑定控件上的数据
- 弹窗提示：正常弹窗和同步弹窗（子线程无法更新 ui）
- 导航到其他页面：正常跳转，携带标志跳转，携带参数跳转
- 存取公共值：封装对 SharedPreferences 的存取操作
- 提供常用对象：gson 转换，上下文 mContext

封装请求操作 API（OK）

为了简化请求操作，将相关代码进行一层封装，所有请求的接口都存在 ApiConfig 中：

```
public class ApiConfig {
    public static final int PAGE_SIZE = 5;
    public static final String BASE_URl = "http://10.0.2.2:8000/";
    public final static String POST_IMG = "http://10.0.2.2:8000/post_img/";    //服务器请求地址
    public final static String PROFILE = "http://10.0.2.2:8000/profile/";    //服务器请求地址
    //用户功能
    public static final String LOGIN = "user/login"; //登录
    public static final String REGISTER = "user/register";//注册
    public static final String USER_INFO = "user/info";//请求用户信息
    public static final String UPDATE_PROFILE = "user/updateProfile";//更新用户头像
    public static final String UPDATE_PWD = "user/updatePwd";//更新用户头像
    public static final String UPDATE_INFO = "user/updateInfo";//更新用户头像
    //帖子功能
    public static final String PUBLISH = "post/publish";//发布新帖子
    public static final String POST_LIST = "post/postList";//发布新帖子
    public static final String POST_Delete = "post/delete";//删除帖子
    public static final String POST_COMMENT = "post/add_command";//提交评论
    public static final String POST_REPLY = "post/add_reply";//提交回复
    public static final String GET_COMMENTS = "post/commends";//获取所有评论
}
```

方便起见这里统一采用 post 请求并对其进行封装，输入参数为请求的 url，参数，回调函数。使用 Api.config().postRequest 来发送请求，封装后请求的步骤就只有封装请求参数，调用请求函数，对响应作处理三个步骤。封装后发送请求的代码标准化为:

```
//发送键值对
FormBody requestBody = new FormBody.Builder()
                        .add("account", account)
                        .add("pwd", pwd)
                        .build();

//或者发送一个对象
```

```
JSONObject json = new JSONObject();
json.put("name",name);//加入各种参数
.....
MediaType JSON = MediaType.parse("application/json;charset=utf-8");
RequestBody requestBody = RequestBody.create(JSON, String.valueOf(json));

Api.config(ApiConfig.REGISTER, requestBody).postRequest(this,new TtitCallback() {
    @Override
    public void onSuccess(final String res) {
    }
    @Override
    public void onFailure(Exception e) {
    }
});
```

九宫格图片的使用(OK)

使用 GitHub 上（https://github.com/jeasonlzy/NineGridView）的开源控件 NineGridView。使用时先在 Application 中初始化 NineGridView 的图片加载器：

```
    NineGridView.setImageLoader(new PicassoImageLoader());
    /** Picasso 加载 */
    private class PicassoImageLoader implements NineGridView.ImageLoader {

        @Override
        public void onDisplayImage(Context context, ImageView imageView, String url) {
            Picasso.with(context).load(url)//
                    .placeholder(R.drawable.ic_default_image)//
                    .error(R.drawable.ic_default_image)//
                    .into(imageView);
        }

        @Override
        public Bitmap getCacheImage(String url) {
            return null;
        }
    }
```

然后再在自己的 Adapter 中初始化 NineGridView 的适配器

```
    ArrayList<ImageInfo> imageInfo = new ArrayList<>();
    List<EvaluationPic> imageDetails = item.getAttachments();
    if (imageDetails != null) {
        for (EvaluationPic imageDetail : imageDetails) {
```

```
            ImageInfo info = new ImageInfo();
            info.setThumbnailUrl(imageDetail.smallImageUrl);
            info.setBigImageUrl(imageDetail.imageUrl);
            imageInfo.add(info);
        }
    }
    holder.nineGrid.setAdapter(new ClickNineGridViewAdapter(context, imageInfo));
```

登录注册和修改信息、密码（OK）

比较常规没什么好说的，主要就是输入的有效性检验，登录完成后将个人信息通过 SharedPreferences 持久化保存在本地方便后续使用。注册主要也是控件布局的美观还有输入的有效性检验。信息和密码的修改也是，要注意的是修改完密码后根据实际应该退出登录让用户重新登录，然后除了输入的有效性检验外要检验原来密码的正确性和两次输入新密码是否相同。

Tablayout 的使用(OK)

Tablayout 使用的是 github 上的项目（https://github.com/H07000223/FlycoTabLayout），有着丰富的动画效果。使用时准备好 ArrayList<Fragment> mFragments 存放每个 tab 的 fragment，用 String[] mTitles = {"首页", "发现", "我的"}用于存每个 tab 的标题，int[] mIconUnselectIds，mIconSelectIds 存放每个 tab 选择与否时展示的图标，用 ArrayList<CustomTabEntity> mTabEntities 存放每个 tab 的信息（即整合上述信息，包括标题，是否选中的图标）。逻辑上，是将 mFragments 放入 viewPager 中，在 tablayout 中绑定点击事件点击不同图标时通过 viewPager.setCurrentItem(position)切换不同页面，在 viewPager 中绑定点击事件通过 commonTabLayout.setCurrentTab(position);使得页面滑动时 tab 栏的图标也发生变化。即通过双方绑定事件来达到两者联动的效果。此外，ViewPager 还需设置一个 adapter 来绑定数据。项目中一个配置 tablayout 的实例代码如下：

```
@Override
protected void initData() {
    mFragments.add(HomeFragment.newInstance());
    mFragments.add(SearchFragment.newInstance());
    mFragments.add(MineFragment.newInstance());
    for (int i = 0; i < mTitles.length; i++) {
        mTabEntities.add(new TabEntity(mTitles[i], mIconSelectIds[i], mIconUnselectIds[i]));
    }
    commonTabLayout.setTabData(mTabEntities);
    commonTabLayout.setOnTabSelectListener(new OnTabSelectListener() {
        @Override
        public void onTabSelect(int position) {
            viewPager.setCurrentItem(position);
        }
```

```
            @Override
            public void onTabReselect(int position) {

            }
        });
        viewPager.setOffscreenPageLimit(mFragments.size());//启动预加载，防止闪退
        viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
            @Override
            public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels)
{

            }

            @Override
            public void onPageSelected(int position) {
                commonTabLayout.setCurrentTab(position);
            }

            @Override
            public void onPageScrollStateChanged(int state) {

            }
        });
        viewPager.setAdapter(new MyPagerAdapter(getSupportFragmentManager(), mTitles,
mFragments));
}
```

图片选择器的使用(OK)

发布帖子和更换头像时需要选择图片，要访问相册或相机，这里使用 PictureSelector 实现。首先当用户点击功能按键时需要弹出一个选择弹窗，这里自定义一个 dialog_deliver_bottom 布局实现，效果如下：

主要是用 3 个 TextView 稍微设置下间隔和圆角，在代码中通过 WindowManager 来添加弹窗，通过 alpha 来调节透明度，showAtLocation 来定位，同时为这 3 个 TextView 绑定点击事件，如果点击取消则关闭弹窗，点击相机/拍照则先请求权限再调用 PictureSelector 最后关闭弹窗。

PictureSelector 的使用比较简单且固定，涉及到的局部相关代码如下：

```
//请求权限
if (ActivityCompat.checkSelfPermission(view.getContext(), Manifest.permission.CAMERA)
        != PackageManager.PERMISSION_GRANTED) {
    // 权限未被授予
    System.out.println("permission not granted detected.");
}
//拍照
PictureSelector.create(DeliverActivity.this)
        .openCamera(PictureMimeType.ofImage())
        .forResult(PictureConfig.CHOOSE_REQUEST);
//相册
PictureSelector.create(DeliverActivity.this)
        //PictureMimeType：全部 ofAll()、图片 ofImage()、视频 ofVideo()、音频 ofAudio()
        .openGallery(PictureMimeType.ofImage())
        .maxSelectNum(adapter.getRestSelectNum())    // 最大图片选择数量 int
        .minSelectNum(1)             // 最小选择数量 int
        .imageSpanCount(4)           // 每行显示个数 int
        .selectionMode(PictureConfig.MULTIPLE)  //多选/单选：MULTIPLE/SINGLE
        .forResult(PictureConfig.CHOOSE_REQUEST);    //结果回调 onActivityResult code
```

在回调中处理返回的图片结果：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    List<LocalMedia> images;
    if (resultCode == RESULT_OK) {
        switch (requestCode) {
            case PictureConfig.CHOOSE_REQUEST:
                // 图片选择结果回调

                images = PictureSelector.obtainMultipleResult(data);
                selectList.addAll(images);

//                  selectList = PictureSelector.obtainMultipleResult(data);
                // LocalMedia 里面返回三种 path
                // 1.media.getPath(); 为原图 path
                // 2.media.getCutPath();为裁剪后 path，需判断 media.isCut();是否为 true
                //  3.media.getCompressPath(); 为压缩后 path，需判断 media.isCompressed();是否为 true
                // 如果裁剪并压缩了，以取压缩路径为准，因为是先裁剪后压缩的
                adapter.setList(selectList);
                adapter.notifyDataSetChanged();
                break;
        }
```

```
    }
}
```

发送帖子的实现（OK）

发送帖子的界面仿微博布局，一个文本输入框输入文字部分，下方有个+的图标点击后弹出选择对话框（详情见图片选择器的使用）使用图片选择器选择图片。点击发送按键后调用 publishBlog 函数，获取帖子的文本内容和图片文件列表，并逐个封装进请求体中，发送服务器，请求成功后弹窗提示并返回原界面。

```
//发布微博帖子
private void publishBlog() {
    String content = text.getText().toString();        //获取帖子的文本
    List<LocalMedia> mediaList = adapter.getMediaList();      //获取需要发送的图片
    //内容判空
    if (content.equals("")) {
        toast.showToastInCenter(getApplicationContext(), "帖子内容不能为空！",
toast.TOAST_UI_QUEUE);
        return ;
    }
    //封装请求
    MultipartBody.Builder builder = new
MultipartBody.Builder().setType(MultipartBody.FORM)
            .addFormDataPart("content", content);
    builder.addFormDataPart("uid", findByKey("id"));
    //添加图片资源
    for (int i = 0; i < mediaList.size(); i++) {
        String path = mediaList.get(i).getPath();     //获取第 i 张图片的路径
        builder = builder.addFormDataPart("pictures", path,
                RequestBody.create(MediaType.parse("application/octet-stream"),     new
File(path)));
    }
    //创建 requestBody
    RequestBody reqbody = builder.build();
    Api.config(ApiConfig.PUBLISH,reqbody).postRequest(DeliverActivity.this,     new
TtitCallback() {
        @Override
        public void onSuccess(String res) {
            Log.v("response:",res);
            //通知 UI 线程
            Message msg = new Message();
            Bundle data = new Bundle();
            data.putString("result", "ok");
            msg.setData(data);
```

```
                mHandler.sendMessage(msg);
            }
            @Override
            public void onFailure(Exception e) {
                showToast("当前网络环境不佳，请稍后再试");
            }
        });
}
```

微博帖子展示界面的实现（OK）

项目中有四个地方需要展示微博帖子：

- 仅展示关注博主的帖子；
- 展示所有人的帖子；
- 展示某个用户的帖子；
- 展示搜索结果的帖子；

展示帖子算是项目中用得最多的一个模块了。由于展示帖子的页面样式大体相同，仅展现的内容不同，故封装一个 ConcernedFragment 用于展示微博帖子，ConcernedFragment 实例化时接收并保存两个入参 command 和 info。Command 包含四种情况（查询所有帖子 all，查询某个用户的帖子 user，查询关注博主的帖子 concerned，查询搜索的帖子 search），四种情况对应的附加信息 info（空值，用户 id，用户 id，搜索词条）

考虑到要实现实际微博的上拉刷新最新评论，下滑加载更多评论，这里采用 smartrefresh 的 RefreshLayout 来实现，每个 ConcernedFragment 实例内置 pageNum 保存当前加载的数据到第几页，pageNum 同时也是请求接口的参数之一。refreshLayout 绑定两个监听事件，其中上拉加载事件中将 pageNum 置 0，同时向服务器发送请求；下滑加载更多事件令 pageNum++同时向服务器发送请求。同时，根据事件的不同调用不同的加载动画。同时需要注意，这里请求完数据后才能更新 ui，所以需要使用 Handler 进行通信。

重写上拉和下拉事件：

```
refreshLayout.setOnRefreshListener(new OnRefreshListener() {
    @Override
    public void onRefresh(RefreshLayout refreshlayout) {
        pageNum = 0;
        getPostList(true);
    }
});
refreshLayout.setOnLoadMoreListener(new OnLoadMoreListener() {
    @Override
    public void onLoadMore(RefreshLayout refreshlayout) {
        pageNum++;
        getPostList(false);
    }
});
```

根据 command 分类分页请求数据：

```
private void getPostList(final boolean isRefresh) {

    //封装请求
    FormBody formBody = new FormBody.Builder()
            .add("pageIndex", String.valueOf(pageNum))
            .add("pageSize", String.valueOf(pageSize))
            .add("command",command)
            .add("info",info)
            .build();
    Api.config(ApiConfig.POST_LIST, formBody).postRequest(getActivity(), new TtitCallback()
{
        @Override
        public void onSuccess(final String res) {
            //调用刷新/加载的动画
            if (isRefresh) {
                refreshLayout.finishRefresh(true);
            } else {
                refreshLayout.finishLoadMore(true);
            }
            PostListResponse response = new Gson().fromJson(res, PostListResponse.class);
            System.out.println(res);
            if (response != null) {
                List<Post> list = response.getContent();
                if (list != null && list.size() > 0) {
                    if (isRefresh) {
                        datas = list;
                    } else {
                        datas.addAll(list);
                    }
                    mHandler.sendEmptyMessage(0);
                } else {
                    if (isRefresh) {
                        showToastSync("暂时无数据");
                    } else {
                        showToastSync("没有更多数据");
                    }
                }
            }
        }

        @Override
        public void onFailure(Exception e) {
            if (isRefresh) {
```

```
                    refreshLayout.finishRefresh(true);
                } else {
                    refreshLayout.finishLoadMore(true);
                }
            }
        });
}
```

Handle 通信：

```
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        switch (msg.what) {
            case 0:
                postAdapter.setDatas(datas);
                postAdapter.notifyDataSetChanged();
                break;
        }
    }
};
```

由于每条帖子的布局相同，这里采用循环列表 RecyclerView 来实现，自定义一个 PostAdapter。定义单个帖子布局 item_layout，自定义 ViewHolder 作为整个帖子并在其中绑定所有控件以便后续代码的书写。重写 onBindViewHolder 方法将数据绑定到控件上，同时绑定相关的点击事件：点击帖子进入帖子详情页、点击头像进入个人控件、绑定删除功能按键。其中删除功能的实现：ConcernedFragment 实例化时如果 command 为 user 且传入的 id 和 findbykey 相同，即说明进入的是当前用户的空间，此时 adapter 处理布局绑定数据时会格外加一个删除按键。

```
    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position)
{
        ViewHolder vh = (ViewHolder) holder;
        Post post = datas.get(position);
        vh.tvName.setText(post.getName());
        vh.tvtime.setText(post.getTime());
        vh.tvtext.setText(post.getText());
        vh.tvComment.setText(String.valueOf(post.getCommentCount()));
        vh.tvCollect.setText(String.valueOf(post.getCollectCount()));
        vh.tvLike.setText(String.valueOf(post.getLikeCount()));
        //加载头像
        Picasso.with(context)
                .load(ApiConfig.PROFILE+post.getProfileURL())
                .transform(new CircleTransform())
                .into(vh.profile);
        //加载九宫格图
```

```
        if(!post.getPics().equals("")){
            String[] urls = post.getPics().split(";");
            ArrayList<ImageInfo> imageInfo = new ArrayList<>();
            for(int i=0;i<urls.length;i++){
                ImageInfo info = new ImageInfo();
                info.setThumbnailUrl(ApiConfig.POST_IMG + urls[i]);
                info.setBigImageUrl(ApiConfig.POST_IMG + urls[i]);
                imageInfo.add(info);
                System.out.println("绑定图片:"+ApiConfig.POST_IMG + urls[i]);
            }
            vh.nineGrid.setAdapter(new NineGridViewClickAdapter(context, imageInfo));
        }
        //点击帖子进入帖子详情页
        vh.item.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent;
                intent = new Intent(context, PostDetailActivity.class);
                intent.putExtra("post", post);
                context.startActivity(intent);
            }
        });
        vh.profile.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent;
                intent = new Intent(context, PersonInfoActivity.class);
                intent.putExtra("uid", post.getUid());
                context.startActivity(intent);
            }
        });
        if(isSelf){
            vh.delete.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Integer id = post.getId();
                    FormBody formBody = new FormBody.Builder()
                            .add("pid", String.valueOf(id))
                            .build();
                    Api.config(ApiConfig.POST_Delete,
formBody).postRequest(context,new TtitCallback() {
                        @Override
                        public void onSuccess(final String res) {    //不理解，这个 res=
响应体?
```

```
                                    Looper.prepare();
                                    Toast.makeText(context, "删除成功",
Toast.LENGTH_SHORT).show();
                                    Looper.loop();
                                }
                                @Override
                                public void onFailure(Exception e) {
                                }
                            });
                        }
                    });
            }
            else{
                vh.delete.setVisibility (View.INVISIBLE);
            }

    }
```

搜索功能的实现（OK）

这个功能倒是相当简单，上一节微博帖子展示界面已经实现了，实例化 ConcernedFragment 时传入参数 command="search"，再加一个搜索框即可。搜索框绑定一个监听事件，每次搜索信息时调用 ConcernedFragment 的 update 功能，将搜索信息 query 赋值给参数 info，请求服务器返回数据更新 ConcernedFragment 即可，代码比较简单略去。

个人空间的设计和实现（OK）

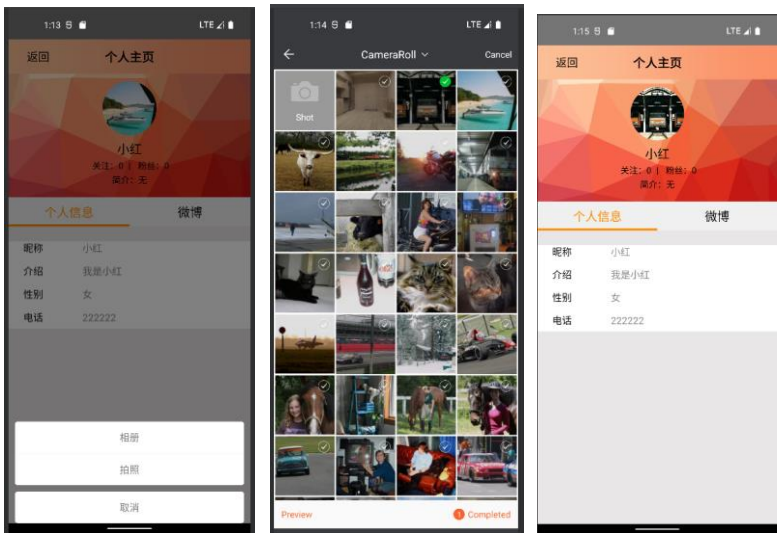
根据实际将微博个人空间的 ui 界面设计如下：

这里使用 CoordinatorLayout 作为顶层布局协调子 View 之间交互，将它与 AppBarLayout 结合即可实现上右图中红框的 toolbar 收缩和扩展的效果，实现方法是内部放置 CollapsingToolbarLayout 并设置 app:layout_scrollFlags="scroll|exitUntilCollapsed"。

然后使用 tablayout 划分一个个人信息栏和一个个人历史微博栏。进入 PersonInfoActivity 时会传入参数 uid，将 uid 作为参数请求服务器查询该用户的信息和所有帖子数据并填充界面，这里注意要用 Handle 进行通信。将 uid 同 findbykey("id)匹配判断用户进入的是不是自己的空间，如果是，实例化 ConcernedFragment 时 adapter 要额外增加一个删除按钮，使得用户可以删除自己的帖子。

更换头像功能（OK）

在个人空间中点击头像可以实现更换头像的功能：



为头像绑定点击事件，点击后唤醒弹窗，接着是前几节图片选择器的流程，得到图片后上传服务器更新用户头像。**这里要特别注意缓存问题**，picasso 有双缓存机制，就是内存缓存和网络缓存，导致就算你给他传新的 url，它也不会去重新访问新的地址上的图片，而是使用缓存的图片。解决方法是请求图片时关闭缓存策略：

```
Picasso.with(PersonInfoActivity.this).load(ApiConfig.PROFILE+findByKey("profile"))
.memoryPolicy(MemoryPolicy.NO_CACHE).networkPolicy(NetworkPolicy.NO_CACHE)
.transform(new CircleTransform()).into(header);
```

同时返回 MineFragment 时要重写 onResume 方法重新加载头像，否则也是使用缓存头像。

二级评论功能的实现（OK）

评论列表准备使用 Recyclerview 实现，实现过程中发现比较困难，一级评论，二级评论，加载评论等元素处理起来较繁琐。调研后发现有个 BaseMultiItemQuickAdapter 非常适合用于这种情况，使用步骤如下：

1.item 元素继承 implements MultiItemEntity 并实现其方法

2. Adapter 继承 BaseMultiItemQuickAdapter

3. Adapter 构造函数中用 addItemType(CommentEntity.TYPE_COMMENT_PARENT, R.layout.item_comment_new);绑定某一类 item 对应的布局

4. Adapter 重写 convert 函数，为某一类 item 绑定控件

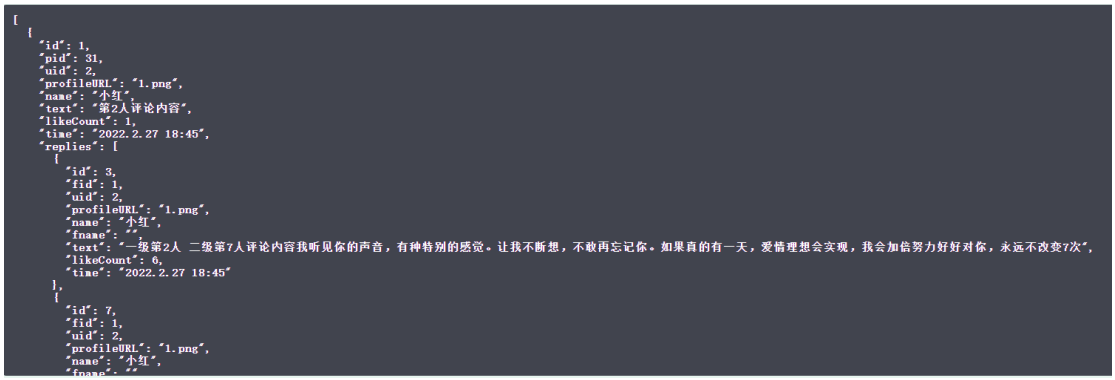
使用 BaseMultiItemQuickAdapter 后便可在一个 adapter 中完成一级评论，二级评论加载评论，空值四种样式的布局了。BaseMultiItemQuickAdapter 内分类：

```
public CommentDialogMutiAdapter(List list) {
    super(list);
    addItemType(CommentEntity.TYPE_COMMENT_PARENT, R.layout.item_comment_new);
    addItemType(CommentEntity.TYPE_COMMENT_CHILD,
R.layout.item_comment_child_new);
    addItemType(CommentEntity.TYPE_COMMENT_MORE,
R.layout.item_comment_new_more);
    addItemType(CommentEntity.TYPE_COMMENT_EMPTY,
R.layout.item_comment_empty);
}

@Override
protected void convert(@NonNull BaseViewHolder helper, MultiItemEntity item) {
    switch (item.getItemType()) {
        case CommentEntity.TYPE_COMMENT_PARENT:
            bindCommentParent(helper, (Comment) item);
            break;
        case CommentEntity.TYPE_COMMENT_CHILD:
            bindCommentChild(helper, (Reply) item);
            break;
        case CommentEntity.TYPE_COMMENT_MORE:
            bindCommonMore(helper, item);
```

```
                break;
            case CommentEntity.TYPE_COMMENT_EMPTY:
                bindEmpty(helper, item);
                break;
        }
}
```

在 PostDetailActivity 中展示帖子详情和所有评论。前端在 getDate()函数中用 datas 接收服务器传来的数据评论，形如：



接收到的数据 datas 是一个 comment 对象数组，其中每个 comment 都有一个 replys 属性，存着一个 reply 对象数组，表示该评论下的所有回复（二级评论）。由于使用的是 BaseMultiItemQuickAdapter，这里需要将所有二级评论提取出来和一级评论同级，即一二级评论都作为一个 item（否则 adapter 会将一个 comment 对象视作一个 item 而不会使用匹配 reply）。在 dataSort()中提取出 reply，给 comment 赋值在 data 中的位置 position，给 reply 赋值第几个一级评论的范围下 position 和在该一级评论的第几个回复 childPosition。然后 commentAdapter.setNewData(data)更新数据。如果为空则 dataSort()中会添加空评论，以便展示“等待第一条评论~~”的提示。此处注意向服务器异步请求完数据后才能调用 dataSort()，这里需要使用 handler 进行通信，异步请求完成后才能通知主进程进行 ui 更新。getDate 请求数据：

```
//向服务器请求评论数据
void getDate(){
    FormBody formBody = new FormBody.Builder()
            .add("pid", String.valueOf(post.getId()))
            .add("pageSize", String.valueOf(40))
            .add("pageIndex", String.valueOf(0))
            .build();
    Api.config(ApiConfig.GET_COMMENTS, formBody).postRequest(this,new TtitCallback()
{
        @Override
        public void onSuccess(final String res) {
            System.out.println(res);
            datas = new Gson().fromJson(res, new TypeToken<List<Comment>>()
{}.getType());
            System.out.println(datas.toString());
            Message message=handler.obtainMessage();
            message.what=1;
```

```
                handler.sendMessage(message);
            }
            @Override
            public void onFailure(Exception e) {
            }
        });
}
```

dataSort()函数：

```
//对数据重新进行排列，让一级评论和二级评论同为 item
private void dataSort(int position) {
    //如果数据为空，添加空评论对象
    if (datas.isEmpty()) {
        data.add(new MultiItemEntity() {
            @Override
            public int getItemType() {
                return CommentEntity.TYPE_COMMENT_EMPTY;
            }
        });
        return;
    }

    if (position <= 0) data.clear();
    int posCount = data.size();//记录位置，赋值给 comment 的 positionCount
    int count = datas.size();//记录评论总数
    for (int i = 0; i < count; i++) {
        if (i < position) continue;

        //先获取一个一级评论
        Comment comment = datas.get(i);
        if (comment == null) continue;
        comment.setPosition(i);//该评论的位置
        posCount += 2;
        //获取该评论的所有回复
        List<Reply> secondLevelBeans = comment.getReplies();
        //子评论为空，添加，跳过
        if (secondLevelBeans == null || secondLevelBeans.isEmpty()) {
            comment.setPositionCount(posCount);
            data.add(comment);
            continue;
        }
        int beanSize = secondLevelBeans.size(); //回复数量
        posCount += beanSize;
        comment.setPositionCount(posCount);
        data.add(comment);
```

```
            //二级评论
            for (int j = 0; j < beanSize; j++) {
                Reply reply = secondLevelBeans.get(j);
                reply.setChildPosition(j);
                reply.setPosition(i);
                reply.setPositionCount(posCount);
                data.add(reply);
            }

//                  //展示更多的 item
//                  if (beanSize <= 18) {
//                      CommentMore moreBean = new CommentMore();
//                      moreBean.setPosition(i);
//                      moreBean.setPositionCount(posCount);
//                      moreBean.setTotalCount(comment.getTotalCount());
//                      data.add(moreBean);
//                  }

        }
}
```

commentAdapter 需要绑定点击事件，对于四类 item：

- 点击了评论：如果点击了点赞按钮，进行点赞处理；如果点击了其他区域，则唤醒输入对话框 inputTextMsgDialog 进行回复；
- 点击了回复：操作同点击了评论；
- 点击了加载更多：请求服务器加载更多评论；
- 点击空值：进行刷新；

唤醒输入对话框时，先通过 view.getTop()获取所点击组件的顶部位置 offsetY，并用 scrollLocation(offsetY)将滚动条移动到 offsetY 处（即正对要回复的评论）。然后实例化一个输入对话框，并为该对话框绑定发送监听事件和关闭监听事件。关闭时滚动条要滚动回原位。发送监听事件处进行发送一条评论或回复。

发送时通过 SharedPreferences 获取发送者信息，根据 isReply 判断为评论还是回复，获取系统时间戳，取出评论信息等，封装成一个 comment 或 reply 对象并发送到服务器并持久化存储到数据库中。然后进行 ui 界面上的评论/回复插入。插入时如果是进行回复，先通过 instanceof 判断点击实体的类别为 comment 还是 reply 并取出评论目标的块序号，块内序号，评论者名称，然后实例化一个 reply 对象并插入到 datas 的对应位置，然后 dataSort 并用 adapter 更新 ui。如果是进行评论则实例化一个 comment 对象然后直接插入进 datas 中然后 dataSort 并用 adapter 更新 ui。

输入对话框和绑定的发送事件：

```
//传入组件 view，是否为回复，item 实体对象，位置
private void initInputTextMsgDialog(View view, final boolean isReply, final MultiItemEntity item, final int position) {
    dismissInputDialog();
```

```
    if (view != null) {
        offsetY = view.getTop();//获取该 view 的顶部
        scrollLocation(offsetY);//滚动条一道该 view 的顶部位置
    }
    if (inputTextMsgDialog == null) {
        inputTextMsgDialog = new InputTextMsgDialog(this, R.style.dialog);
        inputTextMsgDialog.setmOnTextSendListener(new
InputTextMsgDialog.OnTextSendListener() {
            @Override
            public void onTextSend(String msg) {
                //发送回复
                if (position >= 0){
                    //添加二级评论(分为回复 "评论" 和回复 "回复" )
                    String replyUserName = "未知";
                    Integer fid=1;
                    int pos = 0;
                    //回复评论/回复，获取回复对象的名字
                    if (item instanceof Comment) {
                        Comment tmp = (Comment) item;
                        replyUserName = tmp.getName();
                        fid = tmp.getId();
                        positionCount = (int) (tmp.getPositionCount() + 1);
                        pos = (int) tmp.getPosition();
                    } else if (item instanceof Reply) {
                        Reply tmp = (Reply) item;
                        replyUserName = tmp.getName();
                        fid = tmp.getFid();
                        positionCount = (int) (tmp.getPositionCount() + 1);
                        pos = (int) tmp.getPosition();
                    }
                    Reply reply = new Reply();
                    reply.setFname(replyUserName);
                    reply.setFid(fid);
                    reply.setUid(Integer.valueOf(findByKey("id")));
                    reply.setIsReply(isReply ? 1 : 0);
                    reply.setText(msg);
                    reply.setProfileURL(findByKey("profile"));
                    reply.setTime(System.currentTimeMillis());
                    reply.setIsLike(0);
                    reply.setName(findByKey("name"));
                    //发送服务器
                    String param = new Gson().toJson(reply);
                    RequestBody                    requestBody                    =
RequestBody.create(MediaType.parse("application/json"), param);
```

```
                    Api.config(ApiConfig.POST_REPLY,
requestBody).postRequest(PostDetailActivity.this,new TtitCallback() {
                        @Override
                        public void onSuccess(final String res) {
                        }
                        @Override
                        public void onFailure(Exception e) {
                        }
                    });
                    //更新 ui（找到对应的第几个评论插入进它的回复列表中）
                    datas.get(pos).getReplies().add(reply);
                    PostDetailActivity.this.dataSort(0);
                    commentAdapter.notifyDataSetChanged();
                    rv_dialog_lists.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            ((LinearLayoutManager)
rv_dialog_lists.getLayoutManager())
                                    .scrollToPositionWithOffset(positionCount >=
data.size() - 1 ? data.size() - 1
                                            : positionCount, positionCount >=
data.size() - 1 ? Integer.MIN_VALUE : rv_dialog_lists.getHeight());
                        }
                    }, 100);
                }
                //发送一级评论
                else{
                    Comment comment = new Comment();
                    comment.setName(findByKey("name"));//用户名
                    comment.setProfileURL(findByKey("profile"));//头像
                    comment.setTime(System.currentTimeMillis());//发送时间
                    comment.setText(msg);//评论内容
                    comment.setLikeCount(0);//点赞数
                    comment.setPid(post.getId());//帖子 id
                    comment.setUid(Integer.valueOf(findByKey("id")));//评论用户 id
                    //发送服务器
                    String param = new Gson().toJson(comment);
                    RequestBody requestBody =
RequestBody.create(MediaType.parse("application/json"), param);
                    Api.config(ApiConfig.POST_COMMENT,
requestBody).postRequest(PostDetailActivity.this,new TtitCallback() {
                        @Override
                        public void onSuccess(final String res) {
                        }
```

```
                                @Override
                                public void onFailure(Exception e) {
                                }
                            });
                            //更新 ui
                            datas.add(0, comment);
                            PostDetailActivity.this.dataSort(0);
                            commentAdapter.notifyDataSetChanged();
                            rv_dialog_lists.scrollToPosition(0);
                        }
                    }

                    @Override
                    public void dismiss() {
                        //item 滑动到原位
                        scrollLocation(-offsetY);
                    }
                });
            }
            showInputTextMsgDialog();
        }
```

固定底部的评论框(OK)

用 RelativeLayout 布局，然后使用 android:layout_alignParentBottom="true"属性

时间处理(OK)

评论数据在数据库中 time 字段以时间戳的形式存储，前端接收时间戳后用当前时间相减计算出相隔的时间，以此判断后将时间转换成刚刚、X 秒前，X 分钟前，X 小时前等这样的形式，通过 TimeUtils 实现：

```java
public class TimeUtils {

    public static String getRecentTimeSpanByNow(final long millis) {
        long now = System.currentTimeMillis();
        long span = now - millis;
        if (span < 1000) {
            return "刚刚";
        } else if (span < TimeConstants.MIN) {
            return String.format(Locale.getDefault(), format: "%d秒前", ...args: span / TimeConstants.SEC);
        } else if (span < TimeConstants.HOUR) {
            return String.format(Locale.getDefault(), format: "%d分钟前", ...args: span / TimeConstants.MIN);
        } else if (span < TimeConstants.DAY) {
            return String.format(Locale.getDefault(), format: "%d小时前", ...args: span / TimeConstants.HOUR);
        } else if (span < TimeConstants.MONTH) {
            return String.format(Locale.getDefault(), format: "%d天前", ...args: span / TimeConstants.DAY);
        } else if (span < TimeConstants.YEAR) {
            return String.format(Locale.getDefault(), format: "%d月前", ...args: span / TimeConstants.MONTH);
        } else if (span > TimeConstants.YEAR) {
            return String.format(Locale.getDefault(), format: "%d年前", ...args: span / TimeConstants.YEAR);
        } else {
            return String.format("%tF", millis);
        }
    }
}
```

使用时：String time = TimeUtils.getRecentTimeSpanByNow(item.getTime());

效果：