安卓定位应用开发文档

目录

[安卓定位应用开发文档 1](#_Toc199079732)

[1 项目说明 4](#_Toc199079733)

[1.1 技术栈版本 4](#_Toc199079734)

[1.2 核心模块说明 4](#_Toc199079735)

[1.3 主要Activity说明 4](#_Toc199079736)

[2 基础定位功能 5](#_Toc199079737)

[2.1 功能概述 5](#_Toc199079738)

[2.2 核心实现类 5](#_Toc199079739)

[2.2.1 MainActivity.java 5](#_Toc199079740)

[2.2.2 MyApplication.java 6](#_Toc199079741)

[2.3 权限管理系统 6](#_Toc199079742)

[2.3.1 权限申请策略 6](#_Toc199079743)

[2.3.2 权限检查机制 6](#_Toc199079744)

[2.4 定位服务详细配置 6](#_Toc199079745)

[2.4.1 定位精度设置 6](#_Toc199079746)

[2.4.2 定位结果处理 6](#_Toc199079747)

[2.5 地图显示配置 7](#_Toc199079748)

[2.5.1 地图样式设置 7](#_Toc199079749)

[2.5.2 地图交互功能 7](#_Toc199079750)

[2.6 性能优化策略 7](#_Toc199079751)

[2.6.1 定位监控机制 7](#_Toc199079752)

[2.6.2 内存管理 7](#_Toc199079753)

[3 拍照记录功能 8](#_Toc199079754)

[3.1 功能概述 8](#_Toc199079755)

[3.2 拍照流程实现 8](#_Toc199079756)

[3.2.1 相机调用机制 8](#_Toc199079757)

[3.2.2 照片文件管理 8](#_Toc199079758)

[3.3 位置照片数据模型 8](#_Toc199079759)

[3.3.1 LocationPhoto.java 核心设计 8](#_Toc199079760)

[3.3.2 数据存储优化 9](#_Toc199079761)

[3.4 本地存储管理 9](#_Toc199079762)

[3.4.1 LocationPhotoManager.java 设计模式 9](#_Toc199079763)

[3.4.2 数据持久化机制 9](#_Toc199079764)

[3.4.3 照片操作接口 9](#_Toc199079765)

[3.5 历史记录界面 10](#_Toc199079766)

[3.5.1 HistoryActivity.java 界面架构 10](#_Toc199079767)

[3.5.2 地图标记系统 10](#_Toc199079768)

[3.5.3 PhotoFragment.java 列表展示 11](#_Toc199079769)

[3.6 云端上传功能 11](#_Toc199079770)

[3.6.1 LocationPhotoCloud.java 云端数据模型 11](#_Toc199079771)

[3.6.2 上传流程设计 12](#_Toc199079772)

[3.6.3 上传优化策略 13](#_Toc199079773)

[4 周边搜索功能 13](#_Toc199079774)

[4.1 功能概述 13](#_Toc199079775)

[4.2 搜索界面设计 13](#_Toc199079776)

[4.2.1 搜索面板布局 13](#_Toc199079777)

[4.2.2 搜索半径策略 14](#_Toc199079778)

[4.3 POI搜索核心实现 14](#_Toc199079779)

[4.3.1 搜索参数配置 14](#_Toc199079780)

[4.3.2 搜索结果处理 15](#_Toc199079781)

[4.4 搜索结果展示 15](#_Toc199079782)

[4.4.1 地图标记系统 15](#_Toc199079783)

[4.4.2 自定义信息窗口 16](#_Toc199079784)

[4.4.3 智能视野调整 17](#_Toc199079785)

[4.5 导航功能实现 17](#_Toc199079786)

[4.5.1 NavigationActivity.java 架构设计 17](#_Toc199079787)

[4.5.2 多种出行方式支持 17](#_Toc199079788)

[4.5.3 路线搜索与缓存 18](#_Toc199079789)

[4.6 公交路线特殊处理 18](#_Toc199079790)

[4.6.1 最优路线选择 18](#_Toc199079791)

[4.6.2 公交路线可视化 19](#_Toc199079792)

[4.6.3 详细导航指引 19](#_Toc199079793)

[5 位置分享功能 21](#_Toc199079794)

[5.1 功能概述 21](#_Toc199079795)

[5.2 位置分享入口设计 21](#_Toc199079796)

[5.2.1 MapLocationShareActivity.java 架构 21](#_Toc199079797)

[5.2.2 位置信息展示优化 21](#_Toc199079798)

[5.3 文本分享功能 22](#_Toc199079799)

[5.3.1 高德地图链接生成 22](#_Toc199079800)

[5.3.2 分享内容格式化 23](#_Toc199079801)

[5.4 实时位置共享核心架构 23](#_Toc199079802)

[5.4.1 RealTimeLocationShareActivity.java 设计理念 23](#_Toc199079803)

[5.4.2 共享码生成机制 23](#_Toc199079804)

[5.5 LeanCloud数据模型设计 23](#_Toc199079805)

[5.5.1 LocationShare.java 共享会话管理 23](#_Toc199079806)

[5.5.2 LocationData.java 位置数据存储 24](#_Toc199079807)

[5.6 共享流程详细实现 24](#_Toc199079808)

[5.6.1 开始共享流程 24](#_Toc199079809)

[5.6.2 连接对方流程 25](#_Toc199079810)

[5.7 双向位置同步机制 27](#_Toc199079811)

[5.7.1 位置上传定时器 27](#_Toc199079812)

[5.7.2 对方位置查询定时器 27](#_Toc199079813)

[5.8 地图双标记显示 29](#_Toc199079814)

[5.8.1 标记管理策略 29](#_Toc199079815)

[5.8.2 用户体验优化 30](#_Toc199079816)

[5.9 连接管理与资源清理 30](#_Toc199079817)

[5.9.1 断开连接处理 30](#_Toc199079818)

[5.9.2 生命周期管理 31](#_Toc199079819)

[6 项目配置与部署 32](#_Toc199079820)

[6.1 开发环境配置 32](#_Toc199079821)

[6.1.1 Android项目配置 32](#_Toc199079822)

[6.1.2 依赖库配置 33](#_Toc199079823)

[6.2 权限配置详解 33](#_Toc199079824)

[6.2.1 AndroidManifest.xml权限声明 33](#_Toc199079825)

[6.2.2 关键配置说明 34](#_Toc199079826)

[6.3 第三方服务配置 34](#_Toc199079827)

[6.3.1 高德地图API Key配置 34](#_Toc199079828)

[6.3.2 LeanCloud配置 35](#_Toc199079829)

[6.4 FileProvider配置 35](#_Toc199079830)

[6.4.1 文件共享配置 35](#_Toc199079831)

[6.4.2 文件路径配置 35](#_Toc199079832)

[7 总结 35](#_Toc199079833)

1 项目说明

本项目是一个基于高德地图SDK的Android位置应用，提供定位、拍照记录并上传、周边搜索和位置分享、实时共享等功能。应用采用模块化设计，支持本地数据存储和云端数据同步。

1.1 技术栈版本

|  |  |  |
| --- | --- | --- |
| 技术组件 | 版本 | 说明 |
| Android SDK | compileSdk 33,targetSdk 33 | 最低支持 API 21 (Android 5.0) |
| Gradle Plugin | 标准Android应用插件 | 支持Java 8语法特性 |
| 高德地图SDK | AMap3DMap\_10.1.300 | 3D地图显示、定位、搜索 |
| 高德定位SDK | AMapLocation\_6.4.9 | 高精度定位服务 |
| 高德搜索SDK | AMapSearch\_9.7.4 | POI搜索、路线规划 |
| LeanCloud SDK | storage-android:8.2.28 | 云端数据存储 |
| Material Design | 1.4.0 | 现代化UI组件 |
| Gson | 2.10.1 | JSON数据解析 |
| RxJava | 2.2.21 | 异步编程 |

1.2 核心模块说明

基础定位模块：实时获取用户位置，在地图上显示当前位置。

拍照记录模块：拍照并记录位置信息，支持本地存储和云端上传。

周边搜索模块：搜索周边POI，提供路线导航功能。

位置分享模块：支持文本分享和实时位置共享。

1.3 主要Activity说明

MainActivity：应用的核心主界面，集成了高德地图显示和实时定位功能。用户可以在此界面拍照并自动记录位置信息，搜索周边兴趣点如餐厅商店以及其他地点等，点击搜索结果可进行路线导航。同时提供了查看历史照片记录和位置分享的入口，是整个应用功能的总控制台。

HistoryActivity：专门用于管理和展示用户历史拍照记录的界面。下方显示照片列表，上方展示地图，地图上会标记出所有照片的拍摄位置。用户点击照片列表中的任一张照片，地图会自动定位到该照片的拍摄位置并红色高亮显示。还提供了将本地照片上传到云端存储的功能。

PhotoActivity：负责照片拍摄的专用界面。当从主界面进入时执行拍照功能，获取当前位置信息并将照片与位置数据一起保存到本地。当从历史记录进入时则显示已拍摄的照片详情，包括照片内容和地理位置信息，并在小地图上标记拍摄位置。

MapLocationShareActivity：位置分享的准备和展示界面。获取用户当前位置信息后，在地图上清晰标记并显示详细的地址信息。用户可以选择以文本形式分享位置（包含高德地图链接），也可以点击进入实时位置分享功能，为后续的动态位置共享做准备。

RealTimeLocationShareActivity：实现用户间实时位置共享的核心功能界面。用户可以生成6位数字共享码并开始位置广播，其他用户通过输入共享码建立连接。连接成功后双方位置会实时同步显示在地图上，分别用不同标记区分，支持查看对方实时位置变化和地址信息。

NavigationActivity：专业的路线规划和导航指引界面。支持步行、骑行、驾车、公交四种出行方式的路线规划，每种方式用不同颜色在地图上绘制路线。特别针对公交路线提供详细的换乘信息、站点名称、乘坐站数等，帮助用户选择最优出行方案并提供清晰的导航指引。

2 基础定位功能

2.1 功能概述

基础定位功能是整个应用的核心基础，负责获取用户的实时位置信息并在高德地图上进行显示。该功能为后续的拍照记录、周边搜索等功能提供位置数据支撑。

2.2 核心实现类

2.2.1 MainActivity.java

主活动类，承载地图显示和定位功能的核心逻辑。

主要功能：初始化高德地图控件、配置定位服务参数、处理定位权限申请、实时更新位置信息、管理地图交互。

关键代码实现：

*// 定位参数配置*

private void initLocationOption() {

    locationOption = new AMapLocationClientOption();

    locationOption.setLocationMode(AMapLocationClientOption.AMapLocationMode.Hight\_Accuracy);

    locationOption.setNeedAddress(true);           *// 获取详细地址*

    locationOption.setOnceLocation(false);         *// 连续定位*

    locationOption.setInterval(1000);              *// 1秒间隔*

    locationOption.setSensorEnable(true);          *// 开启传感器*

    locationOption.setLocationCacheEnable(false);  *// 关闭缓存*

    locationOption.setGpsFirst(true);              *// 优先GPS*

}

2.2.2 MyApplication.java

应用程序类，负责全局初始化配置。

初始化内容：高德地图隐私合规设置、LeanCloud SDK初始化、全局位置对象管理。

2.3 权限管理系统

2.3.1 权限申请策略

应用采用智能权限申请策略，避免频繁打扰用户：

private static final String[] NEEDED\_PERMISSIONS = new String[]{

    Manifest.permission.ACCESS\_COARSE\_LOCATION,    *// 粗略位置*

    Manifest.permission.ACCESS\_FINE\_LOCATION,      *// 精确位置*

    Manifest.permission.READ\_PHONE\_STATE,          *// 设备状态*

    Manifest.permission.WRITE\_EXTERNAL\_STORAGE,    *// 存储权限*

    Manifest.permission.CAMERA                     *// 相机权限*

};

2.3.2 权限检查机制

首次启动：主动申请必要权限。

权限被拒绝：仅在首次定位失败时才提示。

后台定位：支持Android 10+后台位置权限。

2.4 定位服务详细配置

2.4.1 定位精度设置

高精度模式：同时使用GPS、网络、被动定位。

定位间隔：1秒更新一次，保证实时性。

超时设置：15秒超时，快速响应。

2.4.2 定位结果处理

@Override

public void onLocationChanged(AMapLocation aMapLocation) {

    if (aMapLocation.getErrorCode() == 0) {

*// 定位成功处理*

        currentLocation = aMapLocation;

        updateLocationUI(aMapLocation);

        updateGlobalLocation(aMapLocation);

    } else {

*// 定位失败处理*

        handleLocationError(aMapLocation);

    }

}

2.5 地图显示配置

2.5.1 地图样式设置

定位蓝点：连续跟随模式，实时显示用户位置。

精度圈：半透明蓝色，显示定位精度范围。

缩放级别：默认17级，适合城市道路查看。

控件配置：缩放按钮、定位按钮。

2.5.2 地图交互功能

手势支持：缩放、拖拽、旋转、倾斜。

标记点击：自定义信息窗口。

首次定位：自动移动到当前位置。

位置更新：后续定位一秒刷新一次，以用户所在位置为中心，但不缩放地图。

2.6 性能优化策略

2.6.1 定位监控机制

*// 10秒间隔检查定位服务状态*

private void startLocationRefreshTimer() {

    locationRefreshTimer = new Timer();

    locationRefreshTimer.schedule(new TimerTask() {

        @Override

        public void run() {

            if (!locationClient.isStarted()) {

                restartLocation(); *// 重启定位服务*

            }

        }

    }, LOCATION\_REFRESH\_INTERVAL, LOCATION\_REFRESH\_INTERVAL);

}

2.6.2 内存管理

生命周期管理：在onDestroy中正确释放定位客户端

地图资源：及时回收MapView资源

定时器清理：防止内存泄漏

3 拍照记录功能

3.1 功能概述

拍照记录功能允许用户在当前位置拍摄照片，并自动记录拍摄时的地理位置信息。该功能包含本地存储管理、历史记录查看和云端上传三个核心子功能。

3.2 拍照流程实现

3.2.1 相机调用机制

private void dispatchTakePictureIntent() {

    Intent takePictureIntent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {

        File photoFile = createImageFile(); *// 创建临时文件*

        if (photoFile != null) {

            Uri photoURI = FileProvider.getUriForFile(this,

                "com.example.androidposition.fileprovider", photoFile);

            takePictureIntent.putExtra(MediaStore.EXTRA\_OUTPUT, photoURI);

            startActivityForResult(takePictureIntent, REQUEST\_CAMERA);

        }

    }

}

3.2.2 照片文件管理

文件命名：JPEG\_时间戳\_随机数.jpg

存储路径：应用私有外部存储/Pictures目录

临时处理：拍照完成后立即转换为Base64编码存储

文件清理：编码完成后删除原始文件，节省存储空间

3.3 位置照片数据模型

3.3.1 LocationPhoto.java 核心设计

public class LocationPhoto implements Serializable {

    private double latitude;        *// 纬度*

    private double longitude;       *// 经度*

    private String address;         *// 详细地址*

    private String description;     *// 用户描述*

    private String photoBase64;     *// Base64编码照片*

    private long timestamp;         *// 拍摄时间戳*

*// 100%质量保存，避免压缩*

    private String bitmapToBase64(Bitmap bitmap) {

        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, byteArrayOutputStream);

        byte[] byteArray = byteArrayOutputStream.toByteArray();

        return Base64.encodeToString(byteArray, Base64.DEFAULT);

    }

}

3.3.2 数据存储优化

直接文件读取：从文件路径直接编码，避免Bitmap解码再编码的质量损失。

内存友好：使用Base64存储，避免大量Bitmap对象占用内存。

时间排序：按拍摄时间倒序排列，最新照片优先显示。

3.4 本地存储管理

3.4.1 LocationPhotoManager.java 设计模式

采用单例模式，确保全局唯一的照片管理器：

public class LocationPhotoManager {

    private static LocationPhotoManager instance;

    private final SharedPreferences sharedPreferences;

    private final Gson gson;

    private List<LocationPhoto> photoList;

    public static synchronized LocationPhotoManager getInstance(Context context) {

        if (instance == null) {

            instance = new LocationPhotoManager(context.getApplicationContext());

        }

        return instance;

    }

}

3.4.2 数据持久化机制

存储方式：使用SharedPreferences存储JSON格式的照片列表。

序列化工具：Gson库处理复杂对象序列化。

即时保存：每次添加或删除照片后立即持久化。

数据恢复：应用启动时自动加载历史数据。

3.4.3 照片操作接口

*// 添加照片（支持两种方式）*

public void addPhoto(double lat, double lng, String address, String desc, Bitmap photo);

public void addPhoto(double lat, double lng, String address, String desc, String filePath);

*// 查询操作*

public List<LocationPhoto> getAllPhotos();                    *// 获取所有照片*

public List<LocationPhoto> getNearbyPhotos(double lat, double lng, double radiusKm); *// 附近照片*

public LocationPhoto getPhoto(int index);                     *// 根据索引获取*

*// 管理操作*

public boolean deletePhoto(int index);                        *// 删除照片*

public void updatePhoto(int index, LocationPhoto updatedPhoto); *// 更新照片*

public void clearAllPhotos();                                 *// 清空所有照片*

3.5 历史记录界面

3.5.1 HistoryActivity.java 界面架构

历史记录页面采用上下分屏布局：

上半部分：250dp高度的地图，显示照片拍摄位置

下半部分：Fragment容器，承载照片列表

3.5.2 地图标记系统

private void loadPhotoMarkersToMap() {

    List<LocationPhoto> photos = photoManager.getAllPhotos();

    LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();

    for (int i = 0; i < photos.size(); i++) {

        LocationPhoto photo = photos.get(i);

        LatLng position = new LatLng(photo.getLatitude(), photo.getLongitude());

*// 创建绿色标记，区别于其他功能的标记*

        MarkerOptions markerOptions = new MarkerOptions()

                .position(position)

                .title(photo.getDescription())

                .snippet(photo.getAddress())

                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE\_GREEN));

        Marker marker = aMap.addMarker(markerOptions);

        markerPhotoMap.put(marker, i); *// 建立标记与照片索引的映射*

    }

}

3.5.3 PhotoFragment.java 列表展示

private class PhotoAdapter extends RecyclerView.Adapter<PhotoAdapter.ViewHolder> {

    @Override

    public void onBindViewHolder(@NonNull ViewHolder holder, final int position) {

        LocationPhoto photo = photos.get(position);

*// 显示原图而非缩略图*

        Bitmap bitmap = photo.getPhotoBitmap();

        if (bitmap != null) {

            holder.imageView.setImageBitmap(bitmap);

        }

*// 设置点击事件：在地图上高亮显示*

        holder.itemView.setOnClickListener(v -> {

            ((HistoryActivity) getActivity()).showPhotoOnMap(position);

        });

*// 长按删除功能*

        holder.itemView.setOnLongClickListener(v -> {

            showDeleteConfirmDialog(position);

            return true;

        });

    }

}

3.6 云端上传功能

3.6.1 LocationPhotoCloud.java 云端数据模型

@LCClassName("LocationPhotoCloud")

public class LocationPhotoCloud extends LCObject {

*// 字段定义*

    private static final String FIELD\_USER\_ID = "userId";

    private static final String FIELD\_LATITUDE = "latitude";

    private static final String FIELD\_LONGITUDE = "longitude";

    private static final String FIELD\_ADDRESS = "address";

    private static final String FIELD\_DESCRIPTION = "description";

    private static final String FIELD\_PHOTO\_BASE64 = "photoBase64";

    private static final String FIELD\_TIMESTAMP = "timestamp";      *// 拍摄时间*

    private static final String FIELD\_UPLOAD\_TIME = "uploadTime";   *// 上传时间*

}

3.6.2 上传流程设计

1. 照片选择界面

private void showPhotoSelectionDialog() {

    AlertDialog.Builder builder = new AlertDialog.Builder(this);

    View dialogView = LayoutInflater.from(this).inflate(R.layout.dialog\_photo\_selection, null);

    RecyclerView recyclerView = dialogView.findViewById(R.id.recycler\_view\_photos);

    PhotoSelectionAdapter adapter = new PhotoSelectionAdapter(this, photos,

        (position, photo) -> uploadPhotoToCloud(position, photo));

    recyclerView.setAdapter(adapter);

}

2. 上传确认对话框

private void uploadPhotoToCloud(int position, LocationPhoto photo) {

    int photoSizeKB = (int) (photo.getPhotoBase64().length() \* 0.75 / 1024);

    String message = "确定要将这张照片上传到云端吗？\n\n" +

            "拍摄时间: " + photo.getFormattedDateTime() + "\n" +

            "地址: " + photo.getAddress() + "\n" +

            "图片大小: " + photoSizeKB + " KB\n\n" +

            "⚠️ 较大文件可能需要更长上传时间";

    new AlertDialog.Builder(this)

        .setTitle("确认上传")

        .setMessage(message)

        .setPositiveButton("上传", (dialog, which) -> performUpload(position, photo))

        .show();

}

3. 执行上传操作

private void performUpload(int position, LocationPhoto photo) {

    progressDialog.show(); *// 显示上传进度*

    String userId = "user\_" + UUID.randomUUID().toString().substring(0, 8);

    LocationPhotoCloud cloudPhoto = new LocationPhotoCloud();

*// 设置所有字段...*

    cloudPhoto.saveInBackground().subscribe(new Observer<LCObject>() {

        @Override

        public void onNext(LCObject lcObject) {

            progressDialog.dismiss();

            Toast.makeText(HistoryActivity.this, "照片上传成功！", Toast.LENGTH\_SHORT).show();

        }

        @Override

        public void onError(Throwable e) {

            progressDialog.dismiss();

            Toast.makeText(HistoryActivity.this, "上传失败: " + e.getMessage(), Toast.LENGTH\_LONG).show();

        }

    });

}

3.6.3 上传优化策略

进度提示：显示上传进度对话框，提升用户体验。

错误处理：详细的错误信息提示。

用户确认：显示照片详情供用户确认后再上传。

异步处理：使用RxJava进行异步上传，不阻塞UI线程

4 周边搜索功能

4.1 功能概述

周边搜索功能基于高德地图POI搜索服务，允许用户搜索当前位置周边的兴趣点，并提供多种出行方式的路线导航。该功能支持灵活的搜索半径设置和智能的搜索结果展示。

4.2 搜索界面设计

4.2.1 搜索面板布局

搜索面板采用可折叠设计，默认隐藏，点击"周边搜索"按钮后显示：

<LinearLayout

    android:id="@+id/search\_panel"

    android:layout\_width="match\_parent"

    android:layout\_height="wrap\_content"

    android:orientation="vertical"

    android:background="#FFFFFF"

    android:visibility="gone">

*<!-- 搜索输入框 -->*

    <LinearLayout>

        <EditText android:id="@+id/et\_search" />

        <Button android:id="@+id/btn\_search" />

    </LinearLayout>

*<!-- 搜索半径选择器 -->*

    <RadioGroup android:id="@+id/rg\_search\_radius">

        <RadioButton android:text="500米" />

        <RadioButton android:text="1公里" android:checked="true" />

        <RadioButton android:text="3公里" />

        <RadioButton android:text="5公里" />

        <RadioButton android:text="不限" />

    </RadioGroup>

</LinearLayout>

4.2.2 搜索半径策略

500米：精确范围，适合查找就近设施

1公里：默认选项，平衡搜索范围和结果精度

3公里：中等范围，适合步行或骑行

5公里：较大范围，适合开车

不限：城市内搜索，返回更多结果

4.3 POI搜索核心实现

4.3.1 搜索参数配置

private void searchPOI(String keyword) {

    try {

*// 构建搜索查询*

        PoiSearch.Query query = new PoiSearch.Query(keyword, "", "");

        query.setPageSize(30);  *// 返回30条结果*

        query.setPageNum(0);    *// 第一页*

        PoiSearch poiSearch = new PoiSearch(this, query);

        poiSearch.setOnPoiSearchListener(this);

*// 设置搜索中心点*

        LatLonPoint center = new LatLonPoint(

            currentLocation.getLatitude(),

            currentLocation.getLongitude()

        );

        if (searchRadius > 0) {

*// 周边搜索*

            poiSearch.setBound(new PoiSearch.SearchBound(center, searchRadius));

        } else {

*// 城市内搜索*

            query = new PoiSearch.Query(keyword, "", currentLocation.getCity());

        }

        poiSearch.searchPOIAsyn();

    } catch (AMapException e) {

        Toast.makeText(this, "搜索失败: " + e.getErrorMessage(), Toast.LENGTH\_SHORT).show();

    }

}

4.3.2 搜索结果处理

@Override

public void onPoiSearched(PoiResult poiResult, int errorCode) {

    if (errorCode == AMapException.CODE\_AMAP\_SUCCESS) {

        List<PoiItem> poiItems = poiResult.getPois();

        if (poiItems != null && !poiItems.isEmpty()) {

            clearPOIMarkers(); *// 清除之前的标记*

*// 寻找最近的POI点*

            PoiItem nearestPoi = findNearestPoi(poiItems);

*// 在地图上添加标记*

            addPOIMarkers(poiItems);

*// 调整地图视野显示所有结果*

            adjustMapView(poiItems);

*// 显示最近POI的信息窗口*

            showNearestPoiInfo(nearestPoi);

        }

    }

}

4.4 搜索结果展示

4.4.1 地图标记系统

private void addPOIMarkers(List<PoiItem> poiItems) {

    LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();

    for (PoiItem poiItem : poiItems) {

        LatLng position = new LatLng(

            poiItem.getLatLonPoint().getLatitude(),

            poiItem.getLatLonPoint().getLongitude()

        );

*// 构建详细信息*

        String snippetText = poiItem.getSnippet();

        if (poiItem.getDistance() > 0) {

            snippetText += "\n距离: " + formatDistance(poiItem.getDistance());

        }

*// 创建橙色标记，区别于定位标记*

        MarkerOptions markerOptions = new MarkerOptions()

                .position(position)

                .title(poiItem.getTitle())

                .snippet(snippetText)

                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE\_ORANGE));

        Marker marker = aMap.addMarker(markerOptions);

        poiMarkers.add(marker);

        boundsBuilder.include(position);

    }

}

4.4.2 自定义信息窗口

aMap.setInfoWindowAdapter(new AMap.InfoWindowAdapter() {

    @Override

    public View getInfoWindow(final Marker marker) {

        View infoWindow = getLayoutInflater().inflate(R.layout.custom\_info\_window, null);

        TextView title = infoWindow.findViewById(R.id.title);

        TextView snippet = infoWindow.findViewById(R.id.snippet);

        Button navButton = infoWindow.findViewById(R.id.btn\_navigate);

        title.setText(marker.getTitle());

        snippet.setText(marker.getSnippet());

*// 只在POI标记上显示导航按钮*

        if (poiMarkers.contains(marker)) {

            navButton.setVisibility(View.VISIBLE);

            navButton.setOnClickListener(v -> {

                navigateToPOI(marker);

                marker.hideInfoWindow();

            });

        } else {

            navButton.setVisibility(View.GONE);

        }

        return infoWindow;

    }

});

4.4.3 智能视野调整

private void adjustMapView(List<PoiItem> poiItems) {

    LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();

*// 将当前位置加入边界计算*

    boundsBuilder.include(new LatLng(

        currentLocation.getLatitude(),

        currentLocation.getLongitude()

    ));

*// 添加所有POI位置*

    for (PoiItem item : poiItems) {

        boundsBuilder.include(new LatLng(

            item.getLatLonPoint().getLatitude(),

            item.getLatLonPoint().getLongitude()

        ));

    }

*// 动画移动到合适的视野范围*

    aMap.animateCamera(CameraUpdateFactory.newLatLngBounds(

        boundsBuilder.build(), 200), new AMap.CancelableCallback() {

        @Override

        public void onFinish() {

*// 显示最近POI的信息窗口*

            if (nearestMarker != null) {

                nearestMarker.showInfoWindow();

            }

        }

    });

}

4.5 导航功能实现

4.5.1 NavigationActivity.java 架构设计

导航页面采用分层布局：

地图层：全屏显示，承载路线和标记。

顶部控制栏：返回按钮、标题。

底部信息栏：路线信息、导航模式切换、详细指引。

4.5.2 多种出行方式支持

public enum RouteType {

    DRIVE,  *// 驾车 - 蓝色路线*

    WALK,   *// 步行 - 绿色路线*

    RIDE,   *// 骑行 - 橙色路线*

    BUS     *// 公交 - 紫色路线（地铁为粉红色）*

}

*// 路线颜色配置*

private static final int COLOR\_WALK = Color.parseColor("#4CAF50");      *// 绿色*

private static final int COLOR\_RIDE = Color.parseColor("#FF9800");      *// 橙色*

private static final int COLOR\_DRIVE = Color.parseColor("#2196F3");     *// 蓝色*

private static final int COLOR\_BUS = Color.parseColor("#9C27B0");       *// 紫色*

private static final int COLOR\_SUBWAY = Color.parseColor("#E91E63");     *// 粉红色*

4.5.3 路线搜索与缓存

private Map<RouteType, Object> routeCache = new HashMap<>();

private void searchRouteIfNeeded(RouteType routeType) {

*// 检查缓存*

    if (routeCache.containsKey(routeType)) {

        displayCachedRoute(routeType);

        return;

    }

*// 执行搜索*

    switch (routeType) {

        case DRIVE:

            searchDriveRoute();

            break;

        case WALK:

            searchWalkRoute();

            break;

        case RIDE:

            searchRideRoute();

            break;

        case BUS:

            searchBusRoute();

            break;

    }

}

4.6 公交路线特殊处理

4.6.1 最优路线选择

private BusPath findBestBusPath(List<BusPath> paths) {

    BusPath bestPath = paths.get(0);

    float minWalkDistance = calculateTotalWalkDistance(bestPath);

    for (BusPath currentPath : paths) {

        float currentWalkDistance = calculateTotalWalkDistance(currentPath);

        if (currentWalkDistance < minWalkDistance) {

            minWalkDistance = currentWalkDistance;

            bestPath = currentPath;

        }

    }

    return bestPath; *// 返回步行距离最少的路线*

}

4.6.2 公交路线可视化

private void drawBusRouteOnMap(BusPath busPath) {

    for (BusStep step : busPath.getSteps()) {

*// 绘制步行部分 - 黄色虚线*

        if (step.getWalk() != null && step.getWalk().getDistance() > 50) {

            drawWalkingSegment(step.getWalk(), COLOR\_BUS\_WALK);

        }

*// 绘制公交/地铁线路*

        for (RouteBusLineItem busLine : step.getBusLines()) {

            boolean isSubway = isSubwayLine(busLine.getBusLineName());

            int color = isSubway ? COLOR\_SUBWAY : COLOR\_BUS;

            int width = isSubway ? 20 : 16;

            drawBusLineSegment(busLine, color, width);

            addStationMarkers(busLine, isSubway);

        }

    }

}

4.6.3 详细导航指引

private void showBusRouteDetails(BusPath busPath) {

    StringBuilder details = new StringBuilder();

    details.append("【路线概览】\n");

    details.append("◆ 总步行距离: ").append(formatDistance(calculateTotalWalkDistance(busPath))).append("\n");

    details.append("◆ 换乘次数: ").append(Math.max(0, busPath.getSteps().size() - 1)).append("次\n");

    details.append("─────────────────\n\n");

    int stepIndex = 1;

    for (BusStep step : busPath.getSteps()) {

*// 处理步行段*

        if (step.getWalk() != null && step.getWalk().getDistance() > 50) {

            details.append("【步骤").append(stepIndex++).append("】 🚶 步行")

                   .append("约").append(formatDistance(step.getWalk().getDistance())).append("\n");

        }

*// 处理公交段*

        if (step.getBusLines() != null && !step.getBusLines().isEmpty()) {

            for (RouteBusLineItem busLine : step.getBusLines()) {

                boolean isSubway = isSubwayLine(busLine.getBusLineName());

                details.append("【步骤").append(stepIndex++).append("】 ")

                       .append(isSubway ? "🚇 " : "🚌 ")

                       .append(busLine.getBusLineName()).append("\n");

                if (busLine.getDepartureBusStation() != null) {

                    details.append("   上车: ").append(busLine.getDepartureBusStation().getBusStationName()).append("\n");

                }

                if (busLine.getArrivalBusStation() != null) {

                    details.append("   下车: ").append(busLine.getArrivalBusStation().getBusStationName()).append("\n");

                }

                if (busLine.getPassStationNum() > 0) {

                    details.append("   经过: ").append(busLine.getPassStationNum()).append("站\n");

                }

            }

        }

        details.append("\n");

    }

    busDetailsText.setText(details.toString());

    checkScrollHint(); *// 检查是否需要显示滚动提示*

}

5 位置分享功能

5.1 功能概述

位置分享功能提供两种分享方式：文本分享和实时位置共享。文本分享通过系统分享接口发送位置信息和高德地图链接；实时位置共享基于LeanCloud实现双向位置同步，支持多人实时查看彼此位置。

5.2 位置分享入口设计

5.2.1 MapLocationShareActivity.java 架构

位置分享页面采用简洁的三段式布局：

地图区域：显示当前位置，带有红色标记

位置信息卡片：悬浮显示位置摘要信息

分享按钮区域：文本分享和位置共享两个选项

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity\_map\_location\_share);

*// 从全局Application获取当前位置*

    currentLocation = ((MyApplication) getApplication()).getCurrentLocation();

    if (currentLocation == null) {

        Toast.makeText(this, "无法获取位置信息，请返回重试", Toast.LENGTH\_SHORT).show();

        finish();

        return;

    }

*// 提取位置信息*

    latitude = currentLocation.getLatitude();

    longitude = currentLocation.getLongitude();

    address = currentLocation.getAddress();

    initViews();

    initMap();

    displayLocationInfo();

    addLocationMarker();

}

5.2.2 位置信息展示优化

private void displayLocationInfo() {

    titleTextView.setText("我的位置");

*// 优化地址显示，保持简洁*

    if (address != null && !address.isEmpty()) {

*// 如果地址过长，进行智能截取*

        String displayAddress = address.length() > 30 ?

            address.substring(0, 30) + "..." : address;

        contentTextView.setText(displayAddress);

    } else {

*// 地址为空时显示坐标*

        contentTextView.setText(String.format("%.6f, %.6f", latitude, longitude));

    }

}

5.3 文本分享功能

5.3.1 高德地图链接生成

private void shareLocationAsText() {

    try {

*// 构建高德地图URL，支持一键导航*

        String amapUrl = "https://uri.amap.com/marker?position=" +

                        longitude + "," + latitude +

                        "&name=我的位置&coordinate=gaode";

*// 构建完整分享文本*

        StringBuilder shareText = new StringBuilder();

        shareText.append("我的位置信息\n");

        if (address != null && !address.isEmpty()) {

            shareText.append("地址: ").append(address).append("\n");

        }

        shareText.append("经度: ").append(longitude).append("\n");

        shareText.append("纬度: ").append(latitude).append("\n\n");

        shareText.append("在高德地图中查看: ").append(amapUrl);

*// 创建系统分享意图*

        Intent shareIntent = new Intent(Intent.ACTION\_SEND);

        shareIntent.setType("text/plain");

        shareIntent.putExtra(Intent.EXTRA\_SUBJECT, "位置分享");

        shareIntent.putExtra(Intent.EXTRA\_TEXT, shareText.toString());

*// 显示分享选择器*

        startActivity(Intent.createChooser(shareIntent, "分享位置信息"));

    } catch (Exception e) {

        Toast.makeText(this, "分享位置失败: " + e.getMessage(), Toast.LENGTH\_SHORT).show();

    }

}

5.3.2 分享内容格式化

分享的文本内容包含：

位置描述：清晰的标题标识。

详细地址：高德地图返回的完整地址信息。

坐标信息：精确的经纬度坐标。

地图链接：点击可直接在高德地图中查看并导航。

5.4 实时位置共享核心架构

5.4.1 RealTimeLocationShareActivity.java 设计理念

实时位置共享采用点对点连接模式，通过6位数字共享码建立连接：

public class RealTimeLocationShareActivity extends AppCompatActivity {

    private static final int LOCATION\_INTERVAL = 5000;  *// 5秒更新位置*

    private static final int QUERY\_INTERVAL = 5000;     *// 5秒查询对方位置*

*// 连接状态管理*

    private String userId;              *// 唯一用户标识*

    private String myShareCode;         *// 我的6位共享码*

    private String partnerShareCode;    *// 对方的共享码*

    private boolean isConnected = false;  *// 连接状态*

    private boolean isSharing = false;    *// 分享状态*

*// 定时任务*

    private Timer locationUpdateTimer;        *// 位置上传定时器*

    private Timer partnerLocationQueryTimer;  *// 对方位置查询定时器*

}

5.4.2 共享码生成机制

private String generateShareCode() {

    Random random = new Random();

    int code = 100000 + random.nextInt(900000); *// 生成100000-999999*

    return String.valueOf(code);

}

private boolean isValidShareCode(String code) {

    return code.matches("^\\d{6}$"); *// 验证6位数字格式*

}

5.5 LeanCloud数据模型设计

5.5.1 LocationShare.java 共享会话管理

@LCClassName("LocationShare")

public class LocationShare extends LCObject {

    private static final String FIELD\_SHARE\_CODE = "shareCode";    *// 共享码*

    private static final String FIELD\_USER\_ID = "userId";          *// 用户ID*

    private static final String FIELD\_IS\_ACTIVE = "isActive";      *// 活跃状态*

    private static final String FIELD\_CREATE\_TIME = "createTime";  *// 创建时间*

*// 标记共享会话为活跃状态*

    public void setActive(boolean isActive) {

        put(FIELD\_IS\_ACTIVE, isActive);

    }

}

5.5.2 LocationData.java 位置数据存储

@LCClassName("LocationData")

public class LocationData extends LCObject {

    private static final String FIELD\_SHARE\_CODE = "shareCode";   *// 关联共享码*

    private static final String FIELD\_USER\_ID = "userId";         *// 用户标识*

    private static final String FIELD\_LATITUDE = "latitude";      *// 纬度*

    private static final String FIELD\_LONGITUDE = "longitude";    *// 经度*

    private static final String FIELD\_ADDRESS = "address";        *// 地址信息*

    private static final String FIELD\_TIMESTAMP = "timestamp";    *// 位置时间戳*

}

5.6 共享流程详细实现

5.6.1 开始共享流程

private void startSharing() {

    if (isSharing) return;

*// 1. 生成共享码*

    myShareCode = generateShareCode();

*// 2. 更新UI显示*

    myShareCodeText.setText(myShareCode);

    updateUIToSharingState();

*// 3. 创建云端共享会话*

    createLocationShare();

*// 4. 启动位置上传定时器*

    startUpdateMyLocation();

    isSharing = true;

    Toast.makeText(this, "已开始共享位置，共享码：" + myShareCode, Toast.LENGTH\_SHORT).show();

}

private void createLocationShare() {

    LocationShare locationShare = new LocationShare();

    locationShare.setShareCode(myShareCode);

    locationShare.setUserId(userId);

    locationShare.setActive(true);

    locationShare.setCreateTime(System.currentTimeMillis());

    locationShare.saveInBackground().subscribe(new Observer<LCObject>() {

        @Override

        public void onNext(LCObject lcObject) {

            Log.i(TAG, "共享会话创建成功");

            if (myLocation != null) {

                createInitialLocationData(); *// 创建初始位置数据*

            }

        }

        @Override

        public void onError(Throwable e) {

            Log.e(TAG, "创建共享会话失败: " + e.getMessage());

            runOnUiThread(() ->

                Toast.makeText(RealTimeLocationShareActivity.this,

                    "创建共享失败: " + e.getMessage(), Toast.LENGTH\_SHORT).show());

        }

    });

}

5.6.2 连接对方流程

private void connectWithPartner() {

    partnerShareCode = partnerCodeInput.getText().toString().trim();

*// 1. 输入验证*

    if (TextUtils.isEmpty(partnerShareCode)) {

        Toast.makeText(this, "请输入对方的共享码", Toast.LENGTH\_SHORT).show();

        return;

    }

    if (!isValidShareCode(partnerShareCode)) {

        Toast.makeText(this, "共享码格式错误，应为6位数字", Toast.LENGTH\_SHORT).show();

        return;

    }

    if (partnerShareCode.equals(myShareCode)) {

        Toast.makeText(this, "不能连接自己的共享码", Toast.LENGTH\_SHORT).show();

        return;

    }

*// 2. 检查共享码有效性*

    checkPartnerShareCodeExists();

}

private void checkPartnerShareCodeExists() {

*// 优先查询位置数据表*

    LCQuery<LocationData> dataQuery = LCObject.getQuery(LocationData.class);

    dataQuery.whereEqualTo("shareCode", partnerShareCode);

    dataQuery.orderByDescending("createdAt");

    dataQuery.limit(1);

    dataQuery.findInBackground().subscribe(new Observer<List<LocationData>>() {

        @Override

        public void onNext(List<LocationData> list) {

            if (list != null && !list.isEmpty()) {

                Log.d(TAG, "找到有效的共享码: " + partnerShareCode);

                establishConnection();

            } else {

*// 备用查询共享会话表*

                queryShareCodeTable();

            }

        }

        @Override

        public void onError(Throwable e) {

            Log.e(TAG, "查询失败，尝试备用方案: " + e.getMessage());

            queryShareCodeTable();

        }

    });

}

5.7 双向位置同步机制

5.7.1 位置上传定时器

private void startUpdateMyLocation() {

    if (locationUpdateTimer != null) {

        locationUpdateTimer.cancel();

    }

    locationUpdateTimer = new Timer();

    locationUpdateTimer.schedule(new TimerTask() {

        @Override

        public void run() {

            uploadMyLocation();

        }

    }, 0, LOCATION\_INTERVAL); *// 立即执行，然后每5秒执行一次*

}

private void uploadMyLocation() {

    if (myLocation == null || !isSharing) return;

*// 每次创建新的位置记录，而非更新现有记录*

    LocationData newData = new LocationData();

    newData.setShareCode(myShareCode);

    newData.setUserId(userId);

    newData.setLatitude(myLocation.getLatitude());

    newData.setLongitude(myLocation.getLongitude());

    newData.setAddress(myLocation.getAddress());

    newData.setTimestamp(System.currentTimeMillis());

    newData.saveInBackground().subscribe(new Observer<LCObject>() {

        @Override

        public void onNext(LCObject lcObject) {

            Log.d(TAG, "位置上传成功");

        }

        @Override

        public void onError(Throwable e) {

            Log.e(TAG, "位置上传失败: " + e.getMessage());

        }

    });

}

5.7.2 对方位置查询定时器

private void startQueryPartnerLocation() {

    if (partnerLocationQueryTimer != null) {

        partnerLocationQueryTimer.cancel();

    }

    partnerLocationQueryTimer = new Timer();

    partnerLocationQueryTimer.schedule(new TimerTask() {

        @Override

        public void run() {

            queryPartnerLocation();

        }

    }, 0, QUERY\_INTERVAL);

}

private void queryPartnerLocation() {

    if (!isConnected || TextUtils.isEmpty(partnerShareCode)) return;

*// 查询对方最新位置*

    LCQuery<LocationData> query = LCObject.getQuery(LocationData.class);

    query.whereEqualTo("shareCode", partnerShareCode);

    query.orderByDescending("createdAt");

    query.limit(1);

    query.findInBackground().subscribe(new Observer<List<LocationData>>() {

        @Override

        public void onNext(List<LocationData> list) {

            if (list != null && !list.isEmpty()) {

                LocationData locationData = list.get(0);

*// 在UI线程更新地图*

                new Handler(Looper.getMainLooper()).post(() -> {

                    updatePartnerMarker(

                        locationData.getLatitude(),

                        locationData.getLongitude(),

                        locationData.getAddress()

                    );

                });

            }

        }

        @Override

        public void onError(Throwable e) {

            Log.e(TAG, "查询对方位置失败: " + e.getMessage());

        }

    });

}

5.8 地图双标记显示

5.8.1 标记管理策略

private void updatePartnerMarker(double latitude, double longitude, String address) {

    LatLng position = new LatLng(latitude, longitude);

*// 地址长度优化*

    String displayAddress = address;

    if (displayAddress != null && displayAddress.length() > 30) {

        displayAddress = displayAddress.substring(0, 30) + "...";

    }

    if (partnerMarker == null) {

*// 首次创建对方标记 - 蓝色标记*

        MarkerOptions markerOptions = new MarkerOptions()

                .position(position)

                .title("伙伴位置")

                .snippet(displayAddress)

                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE\_BLUE));

        partnerMarker = aMap.addMarker(markerOptions);

*// 调整地图视野包含双方位置*

        adjustMapToShowBothLocations();

    } else {

*// 更新现有标记位置*

        partnerMarker.setPosition(position);

        partnerMarker.setSnippet(displayAddress);

    }

    partnerMarker.showInfoWindow();

}

private void adjustMapToShowBothLocations() {

    if (myLocation != null && partnerMarker != null) {

        LatLng myLatLng = new LatLng(myLocation.getLatitude(), myLocation.getLongitude());

        LatLng partnerLatLng = partnerMarker.getPosition();

*// 计算中心点*

        double centerLat = (myLatLng.latitude + partnerLatLng.latitude) / 2;

        double centerLng = (myLatLng.longitude + partnerLatLng.longitude) / 2;

        LatLng center = new LatLng(centerLat, centerLng);

*// 移动到中心位置，缩放级别15适合查看两点*

        aMap.moveCamera(CameraUpdateFactory.newLatLngZoom(center, 15));

    }

}

5.8.2 用户体验优化

颜色区分：我的位置（定位蓝点）、对方位置（蓝色标记）。

信息窗口：显示实时地址信息，支持长地址智能截取。

视野控制：可手动调整地图范围同时显示双方位置。

回到我的位置：提供快速回到自己位置的按钮。

5.9 连接管理与资源清理

5.9.1 断开连接处理

private void disconnectFromPartner() {

    if (isConnected) {

*// 停止对方位置查询*

        if (partnerLocationQueryTimer != null) {

            partnerLocationQueryTimer.cancel();

            partnerLocationQueryTimer = null;

        }

*// 重置UI状态*

        connectButton.setEnabled(true);

        disconnectButton.setEnabled(false);

        partnerCodeInput.setEnabled(true);

        partnerCodeInput.setText("");

*// 清除对方标记*

        if (partnerMarker != null) {

            partnerMarker.remove();

            partnerMarker = null;

        }

*// 重置连接状态*

        partnerShareCode = null;

        isConnected = false;

        Toast.makeText(this, "已断开与伙伴的连接", Toast.LENGTH\_SHORT).show();

    }

}

private void stopSharing() {

*// 停止位置上传*

    if (locationUpdateTimer != null) {

        locationUpdateTimer.cancel();

        locationUpdateTimer = null;

    }

*// 标记共享会话为不活跃*

    if (!TextUtils.isEmpty(myShareCode)) {

        deactivateShareSession();

    }

*// 重置UI到初始状态*

    resetUIToInitialState();

*// 重置状态变量*

    isSharing = false;

    myShareCode = null;

    Toast.makeText(this, "已停止位置共享", Toast.LENGTH\_SHORT).show();

}

5.9.2 生命周期管理

@Override

protected void onDestroy() {

    super.onDestroy();

*// 清理定时器*

    if (partnerLocationQueryTimer != null) {

        partnerLocationQueryTimer.cancel();

    }

    if (locationUpdateTimer != null) {

        locationUpdateTimer.cancel();

    }

*// 停止定位服务*

    if (locationClient != null) {

        locationClient.stopLocation();

        locationClient.onDestroy();

    }

*// 销毁地图*

    if (mapView != null) {

        mapView.onDestroy();

    }

*// 标记共享会话为不活跃*

    if (isSharing && !TextUtils.isEmpty(myShareCode)) {

        deactivateShareSessionSync();

    }

}

@Override

public void onBackPressed() {

    if (isSharing) {

*// 共享中退出需要确认*

        new AlertDialog.Builder(this)

                .setTitle("结束共享")

                .setMessage("您正在共享位置，确定要退出并结束共享吗？")

                .setPositiveButton("结束共享并退出", (dialog, which) -> {

                    stopSharing();

                    finish();

                })

                .setNegativeButton("取消", null)

                .show();

    } else {

        super.onBackPressed();

    }

}

6 项目配置与部署

6.1 开发环境配置

6.1.1 Android项目配置

android {

    namespace 'com.example.androidposition'

    compileSdk 33

    defaultConfig {

        applicationId "com.example.androidposition"

        minSdk 21        *// 支持Android 5.0+*

        targetSdk 33     *// 目标API*

        versionCode 1

        versionName "1.0"

    }

*// Java 8支持（LeanCloud要求）*

    compileOptions {

        sourceCompatibility JavaVersion.VERSION\_1\_8

        targetCompatibility JavaVersion.VERSION\_1\_8

    }

*// 本地库文件支持*

    sourceSets {

        main {

            jniLibs.srcDirs = ['libs']

        }

    }

*// 视图绑定启用*

    buildFeatures {

        viewBinding true

    }

}

6.1.2 依赖库配置

dependencies {

*// Android基础库*

    implementation 'androidx.appcompat:appcompat:1.4.0'

    implementation 'com.google.android.material:material:1.4.0'

    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'

    implementation 'androidx.recyclerview:recyclerview:1.2.1'

    implementation 'androidx.cardview:cardview:1.0.0'

*// 高德地图SDK（本地jar包）*

    implementation files('libs/AMap3DMap\_10.1.300\_AMapSearch\_9.7.4\_AMapLocation\_6.4.9\_20250509.jar')

*// JSON处理*

    implementation 'com.google.code.gson:gson:2.10.1'

*// 图片加载*

    implementation 'com.github.bumptech.glide:glide:4.15.1'

    annotationProcessor 'com.github.bumptech.glide:compiler:4.15.1'

*// LeanCloud云服务*

    implementation 'cn.leancloud:storage-android:8.2.28'

    implementation 'cn.leancloud:realtime-android:8.2.28'

    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'

    implementation 'io.reactivex.rxjava2:rxjava:2.2.21'

    implementation 'com.alibaba:fastjson:1.2.83'

    implementation 'com.squareup.okhttp3:okhttp:4.9.1'

}

6.2 权限配置详解

6.2.1 AndroidManifest.xml权限声明

*<!-- 网络相关权限 -->*

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

*<!-- 定位相关权限 -->*

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_BACKGROUND\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />

*<!-- 设备信息权限 -->*

<uses-permission android:name="android.permission.READ\_PHONE\_STATE" />

*<!-- 存储权限 -->*

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android:permission.READ\_EXTERNAL\_STORAGE" />

*<!-- 相机权限 -->*

<uses-permission android:name="android.permission.CAMERA" />

<uses-feature android:name="android.hardware.camera" android:required="false" />

6.2.2 关键配置说明

ACCESS\_BACKGROUND\_LOCATION：Android 10+后台定位必需

CAMERA：拍照功能必需，但设置为非必需硬件特性

FOREGROUND\_SERVICE：支持前台定位服务

6.3 第三方服务配置

6.3.1 高德地图API Key配置

*<!-- 高德地图API Key -->*

<meta-data

    android:name="com.amap.api.v2.apikey"

    android:value="0f0d188bc421764b359c7a7348d10b03" />

*<!-- 高德定位服务 -->*

<service android:name="com.amap.api.location.APSService" />

6.3.2 LeanCloud配置

*<!-- LeanCloud应用配置 -->*

<meta-data

    android:name="LEANCLOUD\_APP\_ID"

    android:value="Bx7KJeHUYH6TOs8w7VG0lTYP-gzGzoHsz" />

<meta-data

    android:name="LEANCLOUD\_APP\_KEY"

    android:value="u7gUzQ1bfGAef7BVqzA5ltt7" />

<meta-data

    android:name="LEANCLOUD\_SERVER\_URL"

    android:value="https://bx7kjehu.lc-cn-n1-shared.com" />

6.4 FileProvider配置

6.4.1 文件共享配置

<provider

    android:name="androidx.core.content.FileProvider"

    android:authorities="com.example.androidposition.fileprovider"

    android:exported="false"

    android:grantUriPermissions="true">

    <meta-data

        android:name="android.support.FILE\_PROVIDER\_PATHS"

        android:resource="@xml/file\_paths" />

</provider>

6.4.2 文件路径配置

<?xml version="1.0" encoding="utf-8"?>

<paths xmlns:android="http://schemas.android.com/apk/res/android">

    <external-files-path

        name="shared\_images"

        path="Pictures/shared\_location" />

</paths>

7 总结

本应用基于高德地图SDK和LeanCloud平台，实现了以下主要功能：

**基础定位**：实现了定位获取、地图展示和位置更新功能。

**拍照记录**：允许用户在当前位置拍照并保存位置信息，包括：拍照功能实现、历史记录的本地存储、照片记录的云端上传。

**周边搜索**：基于高德地图POI搜索API实现，包括：POI搜索功能、搜索结果在地图上的显示、POI详情查看、导航功能

**位置分享**：包含两种分享方式，文本位置分享：通过系统分享功能分享位置文本。实时位置共享：基于LeanCloud实现实时位置共享功能