雾霾探测系统实验报告

1.任务说明

雾霾的频繁出现已严重的影响到人们的出行，对人们的健康造成了重大影响。因此，能在出行前查看雾霾的指数，并采取相应的措施来把雾霾的影响降到最小就显得尤为重要。本系统在分析多种因子的影响下，设计一款手机端雾霾app探测系统。

2.要求

1.定位功能：将定位城市保存在服务器端，并同时显示在客户端。

2.界面设计：包含显示天气和空气质量指数的动态显示。

3.天气详情和空气质量指数：定位后的城市在服务器端获取后，传给天气详情界面，通过所传城市用百度天气api获取对应的天气详情和空气质量指数，并保存在服务器端。

4.完成报告。

3.方案设计

总体设计

本次实验采取前后端分离开发模式。

前端使用Android Studio进行Android原生态开发，在便于使用开发平台提供的Android SDK同时，应用也能更好的适配Android系统。

后端使用SpringBoot+SQL框架，提供客户端与服务端的连接请求和数据的持久化存储。

需求分析

该系统的功能大体可以分为三部分：获取定位功能、获取天气信息功能、前后端通信功能

定位功能：该系统采用较网络定位精确度更高的GPS定位，具体实现由高德地图提供的API完成

天气功能：天气数据由和风天气的API提供，API接口接受的位置参数从系统的定位功能中获取

前后端通信：前后端通信由SpringFrame框架和Retrofit组件实现，数据的持久化存储由SQL和JPA组件实现

4.系统实现

定位功能

通过阅读高德地图提供的开发文档，定位功能的实现主要有以下步骤：

1.在高德地图提供的API开放平台申请Key并下载SDK

2.将下载好的jar包导入到Android项目中

3.获取相关权限：

```
    <!--用于进行网络定位-->

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

```
    <!--用于访问GPS定位-->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <!--用于获取运营商信息，用于支持提供运营商信息相关的接口-->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <!--用于访问wifi网络信息，wifi信息会用于进行网络定位-->
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <!--用于获取wifi的获取权限，wifi信息会用来进行网络定位-->
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <!--用于访问网络，网络定位需要上网-->
    <uses-permission android:name="android.permission.INTERNET"/>
    <!--用于写入缓存数据到扩展存储卡-->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <!--用于申请调用A-GPS模块-->
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
    <!--如果设置了target >= 28 如果需要启动后台定位则必须声明这个权限-->
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <!--如果您的应用需要后台定位权限，且有可能运行在Android Q设备上,并且设置了target>28，必须增加这个权
限声明-->
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

4.定位功能实现

```
    public LocationHelper(Context context, OnLocationResultListener listener) {
        this.mListener = listener;
        try {
            AMapLocationClient.updatePrivacyShow(context, true, true);
            AMapLocationClient.updatePrivacyAgree(context, true);
            mLocationClient = new AMapLocationClient(context);
            initLocationOptions();
        } catch (Exception e) {
            throw new RuntimeException("初始化定位失败", e);
        }
    }

    private void initLocationOptions() {
        AMapLocationClientOption option = new AMapLocationClientOption();
        option.setLocationMode(AMapLocationClientOption.AMapLocationMode.Hight_Accuracy);
        option.setOnceLocationLatest(true);
        option.setNeedAddress(true);
        option.setMockEnable(false);
        option.setLocationCacheEnable(false);
        mLocationClient.setLocationOption(option);
        mLocationClient.setLocationListener(mLocationListener);
    }

    // 开始定位
    public void startLocation() {
        if (mLocationClient != null) {
            mLocationClient.startLocation();
        }
    }
```

天气功能

采用和风天气的Android SDK中的接口来查询天气数据，空气质量数据以及24h内的温度和湿度数据

1.在和风天气API开放平台创建项目和凭据

2.安装和风天气Android SDK

3.初始化QWeather实例并设置Token生成器

4.天气功能实现（以实时天气为例，空气质量和24小时温度湿度数据实现方法类似）

```
    public static void getWeatherData(String locationId, WeatherCallback callback) {
        WeatherParameter parameter = new WeatherParameter(locationId)
                .lang(Lang.ZH_HANS)// 设置语言
                .unit(Unit.METRIC); // 设置单位
        QWeather.instance.weatherNow(parameter, new Callback<WeatherNowResponse>() {
            @Override
            public void onSuccess(WeatherNowResponse response) {
                // 处理成功的响应
                Weather weatherData = new Weather();
                weatherData.setTemp(response.getNow().getTemp() + "°C");
                weatherData.setWeatherInfo(response.getNow().getText());
                callback.onWeatherDataReceived(weatherData);
            }

            @Override
            public void onFailure(ErrorResponse errorResponse) {
                // 处理请求错误
                Log.e(TAG, "Request Error: " + errorResponse.toString());
            }

            @Override
            public void onException(Throwable e) {
                // 处理异常
                Log.e(TAG, "Exception: " + e.getMessage());
            }
        });
    }
```

5.温度湿度折线图的绘制使用了MPAndroidChart工具，初始化LineChart实例并传入从天气API接口获取的每小时温度湿度数据后引入格式xml就可以显示折线图，核心代码如下：

```
public void onSuccess(WeatherHourlyResponse response) {
    if (response.getCode().equals("200")) {
       //记录各项数据
       List<String> xLabels = new ArrayList<>();
       List<Float> temperatureData = new ArrayList<>();
       List<Float> humidityData = new ArrayList<>();
       for (int i = 0; i < 6; i++) {
           xLabels.add(response.getHourly().get(i).getFxTime().substring(11, 16));
temperatureData.add(Float.parseFloat(response.getHourly().get(i).getTemp()));

           humidityData.add(Float.parseFloat(response.getHourly().get(i).getHumidity()));
```

```
        }
        //回调
        callback.onDataReceived(xLabels, setLineChartData(temperatureData, humidityData));
        Log.d(TAG, "X轴标签: " + xLabels + ", 温度数据: " + temperatureData + ", 湿度数据: " +
humidityData);
     }
}
```

前后端通信功能

在客户端使用Retrofit组件和okHttp组件发送并处理http请求

1.创建Restrofit实例:

```
public class ApiClient {
    private static final String BASE_URL = "http://192.168.3.143:8080";//服务器ip地址
    private static Retrofit retrofit = null;

    public static Retrofit getClient() {
        if (retrofit == null) {
            OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
            HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
            logging.setLevel(HttpLoggingInterceptor.Level.BODY);
            httpClient.addInterceptor(logging);

            retrofit = new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create())
                    .client(httpClient.build())
                    .build();
        }
        return retrofit;
    }
}
```

2.创建与服务端交互的实体类InfoData

```
public class InfoData {
    private String deviceId;
    private String province;
    private String city;
    private String district;
    private double latitude;
    private double longitude;
    private String temp;
    private String weatherInfo;
    private String aqi;
    private String pm25;
    private String pm10;
```

```
    //构造函数和getters/setters...
}
```

3.声明与服务端交互的接口类

```
public interface InfoApiService {
    @GET("/api/info/{deviceId}")
    Call<InfoData> getInfo(@Path("deviceId") String deviceId);
    @POST("/api/info")
    Call<InfoResponse<Void>> postInfo(@Body InfoData infoData);
}
```

4.创建回调工具类InfoSF（以查询数据为例）

```
    public static void fetchInfo(String deviceId, InfoCallback callback) {
        Call<InfoData> call = apiService.getInfo(deviceId);
        call.enqueue(new Callback<InfoData>() {
            @Override
            public void onResponse(Call<InfoData> call, Response<InfoData> response) {
                if (response.isSuccessful()) {
                    InfoData infoData = response.body();
                    if (infoData != null) {
                        callback.onSuccess(infoData);
                    } else {
                        Log.e("InfoSF", "Response body is null");
                        callback.onFailure("Response body is null");
                    }
                } else {
                    Log.e("InfoSF", "Response error: " + response.message());
                }
            }

            @Override
            public void onFailure(Call<InfoData> call, Throwable t) {
                Log.e("InfoSF", "Network error: " + t.getMessage());
            }
        });
    }
```

使用SpringBoot框架创建服务端程序

1.创建并连接数据库

```yaml
spring:
  application:
    name: haze-probe-service
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/haze_probe_service
    username: root
    password: zk040802
  jpa:
    hibernate:
      ddl-auto: update
server:
  port: 8080
```

2.创建与数据库交互实体类InfoEntity

```java
@Entity
@Data
@Table(name = "user_info")
@DynamicUpdate
public class InfoEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String deviceId;

    private String province;
    private String city;
    private String district;
    private double latitude;
    private double longitude;
    private String temp;
    private String weatherInfo;
    private String aqi;
    private String pm25;
    private String pm10;
}
```

3.处理客户端发送的请求（以查询为例）

```java
    @GetMapping("/{deviceId}")
    public ResponseEntity<InfoEntity> getInfo(@PathVariable String deviceId) {
        log.info("Get device info by {}", deviceId);
        return infoRepository.findByDeviceId(deviceId)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
```

多线程设计

为了优化用户体验，该系统在处理请求时采用了以ExecutorService为基础的线程池，在获取到位置信息后，获取实时天气数据、空气质量数据、24h温度湿度数据以多线程方式同步进行，减少了用户等待时间，核心代码如下：

```
// 初始化线程池
executorService = Executors.newFixedThreadPool(4);

...

executorService.execute(() -> fetchAirQualityData(location.getLatitude(),
location.getLongitude()));
executorService.execute(() -> fetchWeatherData(locationID));
executorService.execute(() -> fetchLineChartData(locationID));
```

系统界面设计

系统界面设计采取了Android开发传统的xml布局，实际页面如下：

HazeProbe

22°C

阴

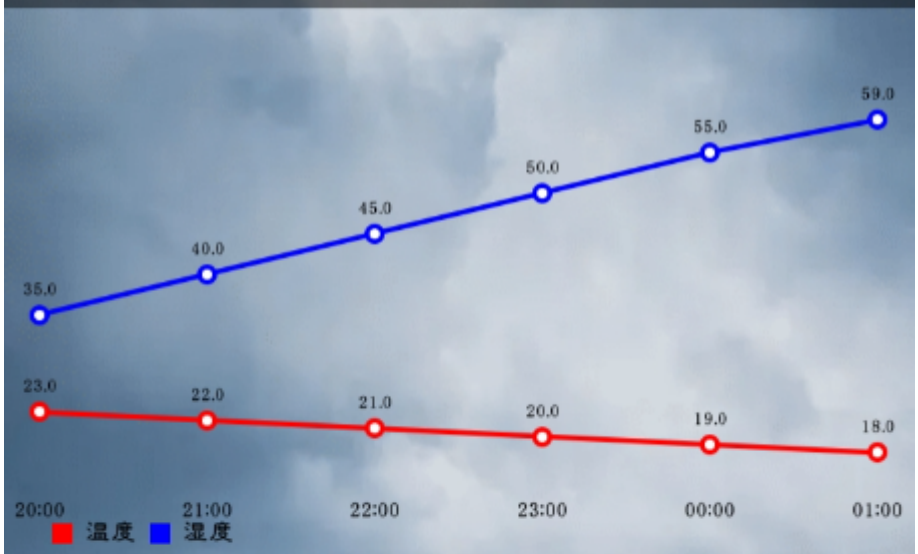
陕西省 西安市 长安区

空气质量

污染指数： 83.0 良

PM2.5： 19.0

PM10： 115.0



获取定位

5.系统运行流程

该系统的入口类为MainActivity，系统在创建时会申请所有需要的权限并进行初始化操作，同时从服务端查询上一次保存的基本数据。

进入应用后，点击右下角的“获取定位”按钮，系统开始调用高德地图的API获取定位信息，获取成功后，调用天气API获取天气数据并渲染到前端页面，同时将更新的数据保存到服务端，这一过程在系统运行的生命周期内始终异步运行，以确保页面展示信息的实时性。

6.心得体会

通过本次实验，我对面向接口编程的理解更加深刻，运用更加熟悉，同时培养了查阅开发文档的编程习惯。这次实验也涉及了很多的组件与框架，通过查阅资料了解组件的接口和使用方法，我的编程能力进一步提高，也让我对前后端分离开发的了解更加深刻。

同时这次的客户端是Android应用程序，通过此次实验，我对Android开发也有了一定的了解。总而言之是一次收获颇丰的实验经历。

但也有许多不足的地方，其中的多线程优化部分，我无法做到定位和获取天气信息同时进行，获取天气信息与定位之间的延时导致系统的使用体验不佳，最后也没能想到解决的办法，恳请老师指导！