**“天气行者”技术报告**

陈朴炎 2021211138

目录

[1 软件介绍 2](#_Toc169128238)

[1.1 软件简介 2](#_Toc169128239)

[1.2 软件功能 2](#_Toc169128240)

[2 传感器编程——LBS 3](#_Toc169128241)

[2.1 位置权限获取 3](#_Toc169128242)

[2.2 GLS 4](#_Toc169128243)

[2.3 LBS——Wifi 定位 7](#_Toc169128244)

[3 网络编程 10](#_Toc169128245)

[3.1 网络权限设置 10](#_Toc169128246)

[3.2 Retrofit 10](#_Toc169128247)

[3.2.1 定义Retrofit服务接口 11](#_Toc169128248)

[3.2.2 定义网络服务类 12](#_Toc169128249)

[3.2.3 定义存放响应信息的类 13](#_Toc169128250)

[3.2.4 使用Retrofit进行网络请求及处理响应 17](#_Toc169128251)

[4 用户界面 18](#_Toc169128252)

[4.1 ViewPager 18](#_Toc169128253)

[4.2 RecyclerView 21](#_Toc169128254)

[5 界面展示 28](#_Toc169128255)

[5.1 本地天气页面 28](#_Toc169128256)

[5.2 搜索页面 30](#_Toc169128257)

# 1 软件介绍

## 1.1 软件简介

天气行者是一款功能简介易用的天气预报应用，它能提供实时准确的天气信息和未来天气预报。无论是日常出行还是计划旅行，天气行者都能为用户提供最贴心的天气服务。

## 1.2 软件功能

（1）本地当前天气查看

天气行者能够获取到手机的位置信息，并将经纬度传给服务器，从而来获取天气信息，其中天气信息包括温度、天气状况。

（2）本地天气预报

天气行者能够展示出本地未来五天每三个小时的天气变化，为用户带来安全可靠的天气预报，让用户能及时应对多变的天气状况。

（3）查看其他城市天气

天气行者还能搜索其他城市的天气信息。当用户滑动到第二页时，将城市英文名称输入至搜索栏后，并点击搜索，就能查询到搜索地的天气情况及预测信息。

（4）背景变换

天气行者能够根据搜索出来的城市天气状况，动态的改变页面的背景图片，在晴朗的时候，背景图片是晴朗的图片，在多云时是多云的图片. . .通过这种变化的背景图案，能够给用户带来沉浸式体验。

# 2 传感器编程——LBS

## 2.1 位置权限获取

首先在AndroidManifest.xml文件里加入位置权限，添加如下内容：

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

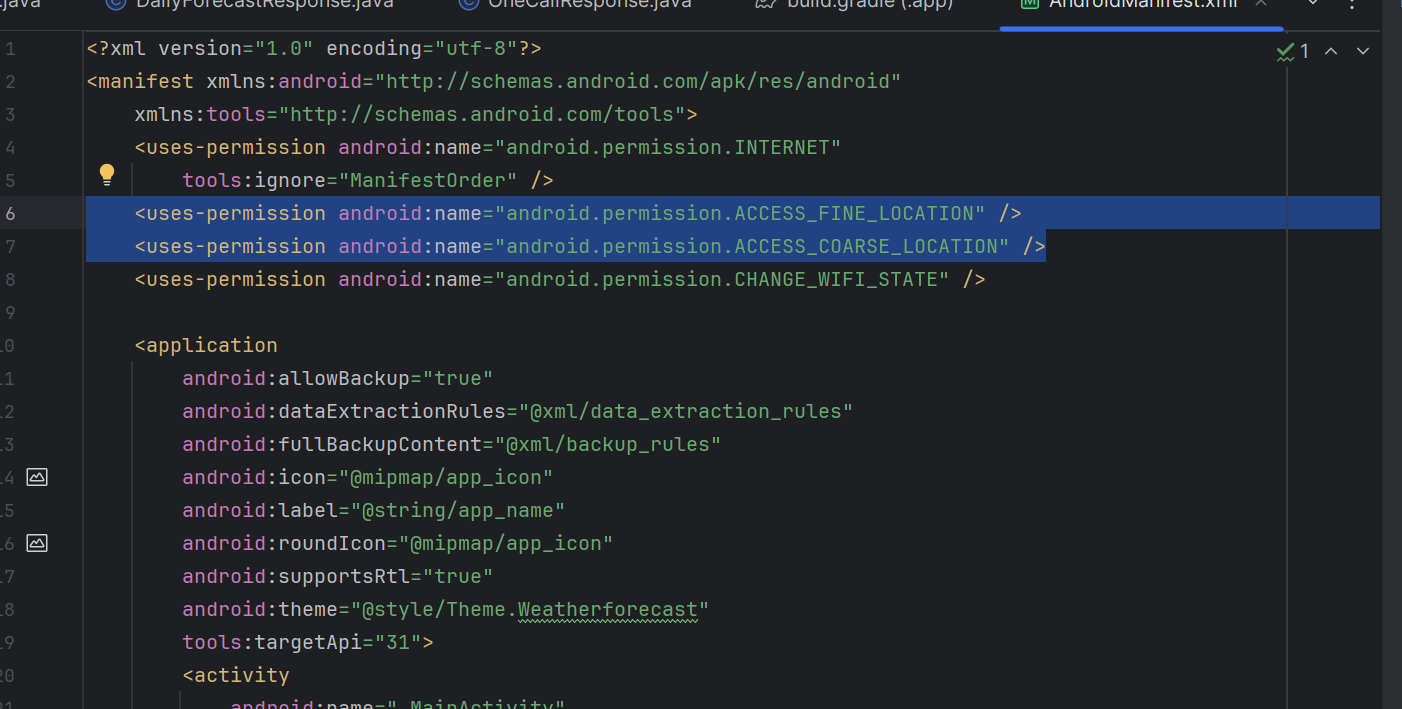


图2-1 添加位置权限示意图

在需要使用到定位服务的地方，先检查是否有位置权限，如下所示

// 检查并请求位置权限  
if (ContextCompat.*checkSelfPermission*(requireContext(), Manifest.permission.*ACCESS\_FINE\_LOCATION*) != PackageManager.*PERMISSION\_GRANTED*) {  
 ActivityCompat.*requestPermissions*(requireActivity(), new String[]{Manifest.permission.*ACCESS\_FINE\_LOCATION*}, *LOCATION\_PERMISSION\_REQUEST\_CODE*);  
 // 未授予权限  
   
   
} else {  
 // 如果权限已经被授予，使用定位服务  
   
}

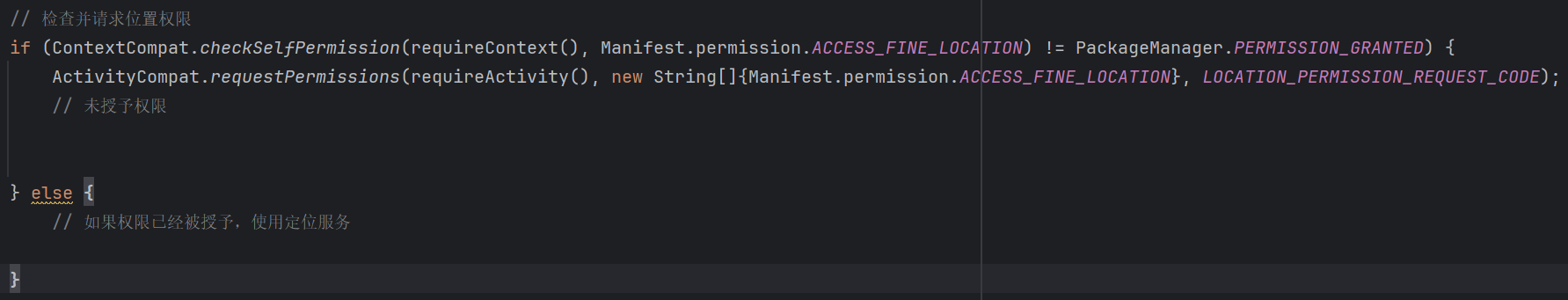
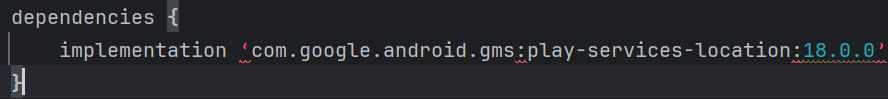


图2-2 动态化检查位置权限示意图

在软件运行时，我们首先通过checkSelfPermission来检查位置权限，如果没有权限的话，那就通过requestPermission来请求用户给予位置权限。

## 2.2 GLS

在这个软件中，为了保证能够获取到位置，我使用了两种获取位置的方式。第一种就是使用Google的位置服务GLS。

Fused Location Provider API是Google Play服务的一部分，它提供了一种方便的方式来访问多种定位服务，并自动选择最适合我们的应用程序的位置提供器。Fused Location Provider API有助于应用程序获取GPS、Wi-Fi、蜂窝网络和传感器等多种定位方式提供的位置数据。 要使用Fused Location Provider API，我们需要在Android项目中添加Google Play服务库。在app下的build.gradle中添加以下依赖项：

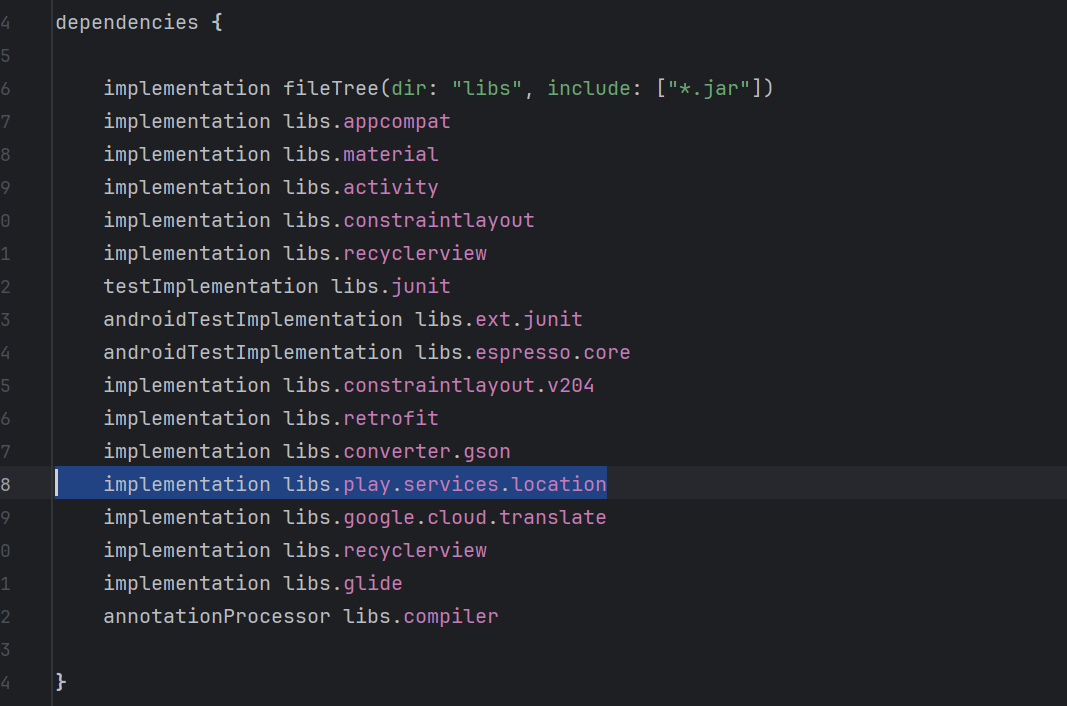


图2-3 在graddle中添加google play 依赖项

在使用Fused Location Provider API获取设备位置之前，需要先检查是否有位置权限。在Android Marshmallow及更高版本中，必须在运行时请求运行时权限。在获取设备位置时，可以使用以下步骤： 建立Fused Location Provider API的客户端，并创建位置请求。 然后监听位置信息。在监听时，我们要处理成功获得监听信息的分支，以及没有成功获取监听信息的分支。

通常第一次使用时，定位可能会比较慢，我们可以先获取最后一次定位的位置信息。

实现处理过程如下：

@SuppressLint({"MissingPermission", "SetTextI18n"})  
private void getCurrentLocationWeather() {  
 cityNameTextView.setText("正在获取位置...");  
 fusedLocationClient.getLastLocation()  
 .addOnSuccessListener(requireActivity(), new OnSuccessListener<Location>() {  
 @SuppressLint("SetTextI18n")  
 @Override  
 public void onSuccess(Location loc) {  
 if (loc != null) {  
 lat = loc.getLatitude();  
 lon = loc.getLongitude();  
 isLocated = true;  
 fetchWeatherData(lat, lon);  
 location = loc;  
 } else {  
 cityNameTextView.setText("Beijing");  
 fetchWeatherData(lat, lon);  
 Toast.*makeText*(getActivity(), "Unable to get location", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @SuppressLint("SetTextI18n")  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 locationWeatherTextView.setText("Failed to get location: " + e.getMessage());  
 Toast.*makeText*(getActivity(), "Failed to get location", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
}



图2-4 获取位置示意图

一般通过这个方法就能成功获取到定位信息了，但是安全起见，我还是加了一层保险——使用LocationManager，请看2.3节。

## 2.3 LBS——Wifi 定位

android.location 是 Android SDK 中的一个包，用于提供与位置服务相关的类和接口。它允许开发者获取设备的地理位置，并使用 GPS、网络等不同的定位提供器来确定位置。

几个重要的类如下:

LocationManager 管理 Android 用户位置服务信息，提供确定用户位置的 API，通过这个类可以实现定位、跟踪和目标趋近等功能

Location 数据类，包含位置信息, 它用来表示地理位置，包含纬度、经度、高度、速度、方向等信息。

LocationListener，一个接口，用于接收位置更新和状态变化的回调。

Geocoder，用于进行地理编码和反向地理编码，即通过地址获取经纬度或通过经纬度获取地址。

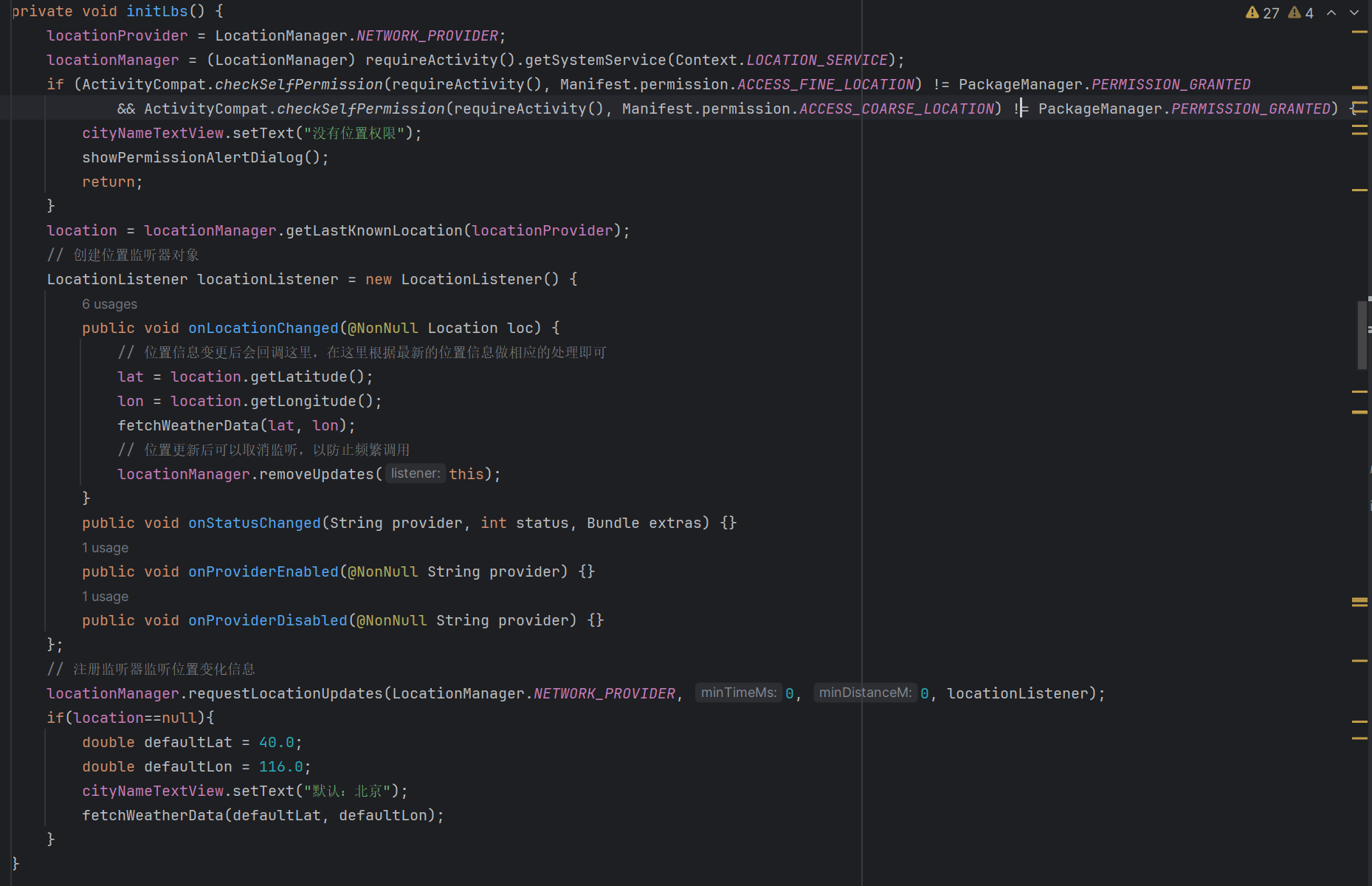
在使用android.location库时，我们需要先获取系统服务，这里我以网络提供商获取位置信息为例。

首先我们要定义一个LocationManager类，然后通过当前Activity来获得系统上下文的位置服务。在使用位置服务之前，我们需要检查一下用户有没有给我们位置权限，不然可能会对一个空指针操作。由于第一次获取定位的时候，时间总是很长，所以我们一般先获取上一次的位置信息，并传回给Location类。

这样我们就可以获得一个位置信息，但是这个位置信息是静态的，只有在软件刚启动的时候才能获取一次。为了获得动态的位置信息，我们需要开启一个监听器，当位置发生变化的时候，要将位置读取出来。

为了让软件不那么消耗资源，我们可以在位置变化过一次之后就将监听器释放掉，以此节省电量。

具体过程如下：

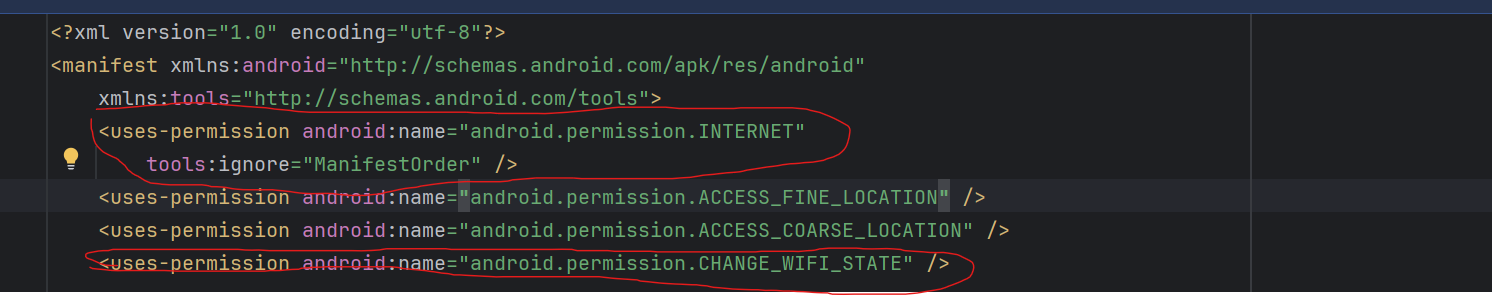


private void initLbs() {  
 locationProvider = LocationManager.*NETWORK\_PROVIDER*;  
 locationManager = (LocationManager) requireActivity().getSystemService(Context.*LOCATION\_SERVICE*);  
 if (ActivityCompat.*checkSelfPermission*(requireActivity(), Manifest.permission.*ACCESS\_FINE\_LOCATION*) != PackageManager.*PERMISSION\_GRANTED* && ActivityCompat.*checkSelfPermission*(requireActivity(), Manifest.permission.*ACCESS\_COARSE\_LOCATION*) != PackageManager.*PERMISSION\_GRANTED*) {  
 cityNameTextView.setText("没有位置权限");  
 showPermissionAlertDialog();  
 return;  
 }  
 location = locationManager.getLastKnownLocation(locationProvider);  
 // 创建位置监听器对象  
 LocationListener locationListener = new LocationListener() {  
 public void onLocationChanged(@NonNull Location loc) {  
 // 位置信息变更后会回调这里，在这里根据最新的位置信息做相应的处理即可  
 lat = location.getLatitude();  
 lon = location.getLongitude();  
 fetchWeatherData(lat, lon);  
 // 位置更新后可以取消监听，以防止频繁调用  
 locationManager.removeUpdates(this);  
 }  
 public void onStatusChanged(String provider, int status, Bundle extras) {}  
 public void onProviderEnabled(@NonNull String provider) {}  
 public void onProviderDisabled(@NonNull String provider) {}  
 };  
 // 注册监听器监听位置变化信息  
 locationManager.requestLocationUpdates(LocationManager.*NETWORK\_PROVIDER*, 0, 0, locationListener);  
 if(location==null){  
 double defaultLat = 40.0;  
 double defaultLon = 116.0;  
 cityNameTextView.setText("默认：北京");  
 fetchWeatherData(defaultLat, defaultLon);  
 }  
}

# 3 网络编程

## 3.1 网络权限设置

在使用网络服务之前，我们需要设置一下网络权限。如下所示，加入两行权限信息。



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

## 3.2 Retrofit

Retrofit 是一个用于 Android 和 Java 的类型安全的 HTTP 客户端库，它简化了与 RESTful Web 服务进行通信的过程。它构建在 OkHttp 库之上，提供了一个简洁而强大的 API，使得处理网络请求变得更加容易和直观。

在这个软件实现中，我是用Retrofit库来处理http请求、连接、响应。

### 3.2.1 定义Retrofit服务接口

为了能够创建Retrofit实例，我们需要为每个url资源路径创建一个服务接口，该接口需要将查询内容放在回调函数中。

比如下面这个WeatherApi接口，是使用 Retrofit 创建的，它定义了一些 HTTP 请求方法，用于获取天气数据。

这个接口中的方法使用了 Retrofit 的注解 @GET，它指定了每个方法对应的 HTTP 请求类型（GET 请求），并指定了请求的相对路径。例如，@GET("weather") 指定了获取当前天气数据的请求路径是 weather。

接口中的方法使用了 @Query 注解，用于指定请求的查询参数。例如，@Query("q") String city 表示使用 q 参数来指定城市名，@Query("appid") String apiKey 表示使用 appid 参数来指定 API 密钥。

通过定义这些方法，我们可以通过调用接口的方法来发起 HTTP 请求，从而获取天气数据。使用 Retrofit 创建的这个接口提供了一个更加简洁和方便的方式来处理网络请求，使得代码更加清晰易读。

public interface WeatherApi {  
 @GET("weather")  
 Call<WeatherResponse> getCurrentWeather(  
 @Query("q") String city,  
 @Query("appid") String apiKey,  
 @Query("units") String units,  
 @Query("lang") String lang  
 );  
  
 @GET("weather")  
 Call<WeatherResponse> getCurrentWeatherByLocation(  
 @Query("lat") double lat,  
 @Query("lon") double lon,  
 @Query("appid") String apiKey,  
 @Query("units") String units,  
 @Query("lang") String lang  
 );  
 // 获取16天的每日天气预报数据  
 @GET("forecast")  
 Call<WeatherForecastResponse> getFiveDayThreeHourForecast(  
 @Query("lat") double lat,  
 @Query("lon") double lon,  
 @Query("appid") String apiKey,  
 @Query("units") String units,  
 @Query("lang") String language  
 );  
 @GET("forecast")  
 Call<WeatherForecastResponse> getFiveDayThreeHourForecastByCityName(  
 @Query("q") String cityName,  
 @Query("appid") String apiKey,  
 @Query("units") String units,  
 @Query("lang") String lang  
 );  
  
}

### 3.2.2 定义网络服务类

我们需要定义一个网络服务类，这个类用来保存请求服务的URL、以及请求服务的接口，并为这些接口创建Retrofit实例。

在本次软件实现中，我们需要从下面这几条url资源路径获得信息。

"https://api.openweathermap.org/data/2.5/"

"https://api.openweathermap.org/data/3.0/"

"http://api.openweathermap.org/geo/1.0/"

在定义好url后，需要定义内部的接口属性，以及获取接口的方式。



### 3.2.3 定义存放响应信息的类

我们要从服务器获取的数据格式如下所示

{

"cod": "200",

"message": 0,

"cnt": 40,

"list": [

{

"dt": 1661871600,

"main": {

"temp": 296.76,

"feels\_like": 296.98,

"temp\_min": 296.76,

"temp\_max": 297.87,

"pressure": 1015,

"sea\_level": 1015,

"grnd\_level": 933,

"humidity": 69,

"temp\_kf": -1.11

},

"weather": [

{

"id": 500,

"main": "Rain",

"description": "light rain",

"icon": "10d"

}

],

. . . . . .

为了正确接收这些数据，我们需要将这个结构体给提取出来。我们可以创建一个类，来承载这些信息。从消息的内容可以看到，有字符串cod、一个list结构，里面的元素是一个表类型，键值有dt、main、weather、clouds等等. . .

其中，weather还是一个列表，main又是一个字典。我们可以在承载数据的结构体中创建内部类来代表main和weather字段。如下所示：  
 public class WeatherForecastResponse {  
 @SerializedName("list")  
 private List<WeatherForecastItem> forecastList;  
  
 public List<WeatherForecastItem> getForecastList() {  
 return forecastList;  
 }  
  
 public static class WeatherForecastItem {  
 @SerializedName("dt\_txt")  
 private String dateText;  
  
 @SerializedName("main")  
 private Main main;  
  
 @SerializedName("weather")  
 private List<Weather> weather;  
  
 public String getDateText() {  
 return dateText;  
 }  
  
 public Main getMain() {  
 return main;  
 }  
  
 public List<Weather> getWeather() {  
 return weather;  
 }  
  
 public static class Main {  
 @SerializedName("temp")  
 private double temp;  
  
 @SerializedName("temp\_min")  
 private double tempMin;  
  
 @SerializedName("temp\_max")  
 private double tempMax;  
  
 @SerializedName("pressure")  
 private int pressure;  
  
 @SerializedName("humidity")  
 private int humidity;  
  
 public double getTemp() {  
 return temp;  
 }  
  
 public double getTempMin() {  
 return tempMin;  
 }  
  
 public double getTempMax() {  
 return tempMax;  
 }  
  
 public int getPressure() {  
 return pressure;  
 }  
  
 public int getHumidity() {  
 return humidity;  
 }  
 }  
  
 public static class Weather {  
 @SerializedName("description")  
 private String description;  
 @SerializedName("icon")  
 private String icon;  
  
 public String getIcon(){ return icon;}  
  
 public String getDescription() {  
 return description;  
 }  
 }  
 }  
}

在这个类中使用了 @SerializedName 注解来指定 JSON 中的字段名与 Java 类中的属性名之间的映射关系。

在 Retrofit 中，当发起网络请求并获取到服务器响应时，Retrofit 会自动将 JSON 数据转换为 Java 对象。这个类就是为了定义这个转换过程中的数据结构。

具体来说，当我们使用 Retrofit 发起请求时，例如调用 WeatherApi 接口中的方法来获取天气预报数据，Retrofit 会发送网络请求，并将服务器响应的 JSON 数据转换为 WeatherForecastResponse 类的对象。

可以通过调用 getForecastList() 方法来获取预报列表，每个 WeatherForecastItem 对象表示一个时间点的天气预报信息，包括日期、温度、气压、湿度等信息。其中，Main 类和 Weather 类是 WeatherForecastItem 的内部类，用来表示天气预报中的主要信息和天气情况。

### 3.2.4 使用Retrofit进行网络请求及处理响应

当我们设计好接口、承载信息的类之后，就可以开始进行网络请求以及响应处理了，我们需要进行以下几步来使用http服务：

创建 Retrofit 实例：我们可以通过 Retrofit 接口 WeatherApi 中定义的获取天气预报数据的方法来发送请求，比如使用WeatherApi中自定义的getFiveDayThreeHourForecastByCityName方法，这个方法会返回一个 Call<WeatherForecastResponse> 对象，表示了一个异步的网络请求。

然后发起网络请求，我们需要调用 enqueue() 方法来异步执行网络请求，传入一个 Callback<WeatherForecastResponse> 对象作为回调。在这个回调中，定义了请求成功和请求失败时的处理逻辑。

请求成功的处理逻辑：在 onResponse() 方法中，首先通过 response.isSuccessful() 方法判断请求是否成功，并且确保响应的主体部分不为空。如果请求成功且响应主体不为空，就从响应中获取到天气预报数据，

具体实现过程如下：

private void fetchForecastDataLoadRecyclerView(double lat, double lon) {  
 weatherApi.getFiveDayThreeHourForecast(lat, lon, *API\_KEY*, "metric", "zh\_cn").enqueue(new Callback<WeatherForecastResponse>() {  
 @Override  
 public void onResponse(@NonNull Call<WeatherForecastResponse> call, @NonNull Response<WeatherForecastResponse> response) {  
 if (response.isSuccessful() && response.body() != null) {  
 WeatherForecastResponse forecastResponse = response.body();  
 List<WeatherForecastResponse.WeatherForecastItem> forecastList = forecastResponse.getForecastList();  
 HourlyWeatherAdapter adapter = new HourlyWeatherAdapter(forecastList);  
 hourlyWeatherTextView.setText("天气预报");  
 recyclerView.setAdapter(adapter);  
 }  
 }  
  
 @Override  
 public void onFailure(Call<WeatherForecastResponse> call, Throwable t) {  
  
 }  
 });  
}

如果想要通过retrofit查询其他的内容，也可以用上面的这几步。

# 4 用户界面

## 4.1 ViewPager

在我的软件中，我想给用户一个丝滑体验，不仅能看用户当前地区的天气状况，也可以在另一个界面中查询其他地区的天气情况，因此我使用了ViewPager作为布局容器。

ViewPager的作用如下：

实现多页面切换：ViewPager 允许用户在同一个屏幕上水平滑动以切换不同的页面，可以用于展示多个相关或独立的内容页面，例如轮播图、引导页面、选项卡视图等。

提供流畅的滑动体验：ViewPager 提供了内置的手势支持，用户可以通过手指滑动或者程序触发的方式来切换页面，实现流畅的滑动体验。

支持自定义页面布局：每个页面可以包含自定义的布局，因此可以实现各种样式的页面内容，包括文本、图像、列表、网格等。

内存优化：ViewPager 会预加载当前页面的相邻页面，以提高页面切换的流畅度，同时在需要时释放不再需要的页面，从而优化内存使用。

与 Fragment 结合使用：ViewPager 可以与 Fragment 结合使用，每个页面对应一个 Fragment 实例，使得页面内容更加模块化，便于管理和维护

在界面设计中，我想让用户能够看到导航栏，因此使用了一个Tablayout，并在导航栏下面使用了ViewPager来容纳两个页面。

在界面实现中，先对layout中的activity\_main.xml修改，如下所示：

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
  
 <com.google.android.material.tabs.TabLayout  
 android:id="@+id/tabLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:tabMode="fixed" />  
  
 <androidx.viewpager.widget.ViewPager  
 android:id="@+id/viewPager"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />  
</LinearLayout>

然后在MainActivity.java中，需要先创建一个管理ViewPager的内部类，它根据ViewPager中的Fragment的位置来展示页面。这个内部类要继承自FragmentPagerAdapter，并重写里面的方法。而在MainActicity中，我们需要重写Oncreate函数，在创建主界面时，要把TabLayout和ViewPager给初始化出来。如下所示：

public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 ViewPager viewPager = findViewById(R.id.*viewPager*);  
 @SuppressLint({"MissingInflatedId", "LocalSuppress"}) TabLayout tabLayout = findViewById(R.id.*tabLayout*);  
  
 viewPager.setAdapter(new WeatherPagerAdapter(getSupportFragmentManager()));  
 tabLayout.setupWithViewPager(viewPager);  
 }  
  
 private class WeatherPagerAdapter extends FragmentPagerAdapter {  
  
 public WeatherPagerAdapter(FragmentManager fm) {  
 super(fm, *BEHAVIOR\_RESUME\_ONLY\_CURRENT\_FRAGMENT*);  
 }  
  
 @NonNull  
 @Override  
 public Fragment getItem(int position) {  
 if (position == 0) {  
 return new LocationWeatherFragment();  
 } else {  
 return new SearchWeatherFragment();  
 }  
 }  
  
 @Override  
 public int getCount() {  
 return 2;  
 }  
  
 @Nullable  
 @Override  
 public CharSequence getPageTitle(int position) {  
 if (position == 0) {  
 return "本地天气";  
 } else {  
 return "搜索其他城市天气";  
 }  
 }  
 }  
}

## 4.2 RecyclerView

RecyclerView是官方在5.0之后新添加的控件，用于展示大量数据列表的强大组件，它是 ListView 和 GridView 的进化版，提供了更灵活、高效的方式来展示和管理数据列表。相比于传统的 ListView，RecyclerView 具有更好的性能和可扩展性，能够适应各种不同的布局需求和交互方式。

RecyclerView 使用布局管理器来确定列表中子项的排列方式。Android 提供了多种预定义的布局管理器，如 LinearLayoutManager（线性布局管理器）、GridLayoutManager（网格布局管理器）、StaggeredGridLayoutManager（瀑布流布局管理器）等，也可以自定义布局管理器以满足特定需求。

为了提高列表的性能，RecyclerView 使用了 ViewHolder 模式。ViewHolder 将每个子项的视图缓存起来，避免频繁地调用 findViewById() 方法，从而提高了列表的滑动流畅性。

RecyclerView 支持自定义分割线，用于在列表项之间添加装饰。另外，还可以通过 ItemAnimator 来实现列表项的添加、移除等动画效果，为用户提供更好的交互体验。

RecyclerView 支持局部刷新，可以精确地更新列表中的某个子项，而不需要重新刷新整个列表。

RecyclerView 提供了丰富的点击和滑动事件处理方式，包括单击、长按、拖拽、滑动删除等，可以根据需求来处理用户的交互操作。

在本次软件实现中，我使用了RecyclerView来保存天气预测数据。因为我想要将每一行都是以日期、时间、天气状况图标、温度的格式排列天气预测数据。由于预测数据的元素数时会变化的，因此用RecyclerView可以很好的动态维护这个预测数据。

使用RecyclerView时，需要执行以下几个步骤：

1. 在build.gradle文件中添加依赖

2. 在布局文件中添加RecyclerView布局

3. 创建RecyclerView里每个元素的布局文件

4. 创建一个继承自RecyclerView.Adapter的类来管理RecyclerView。

5. 在4.步骤中的类中创建一个继承了RecyclerView.ViewHolder类的内部类，用来动态绑定布局容器。

6. 重写4.创建的类中的一些必要的函数

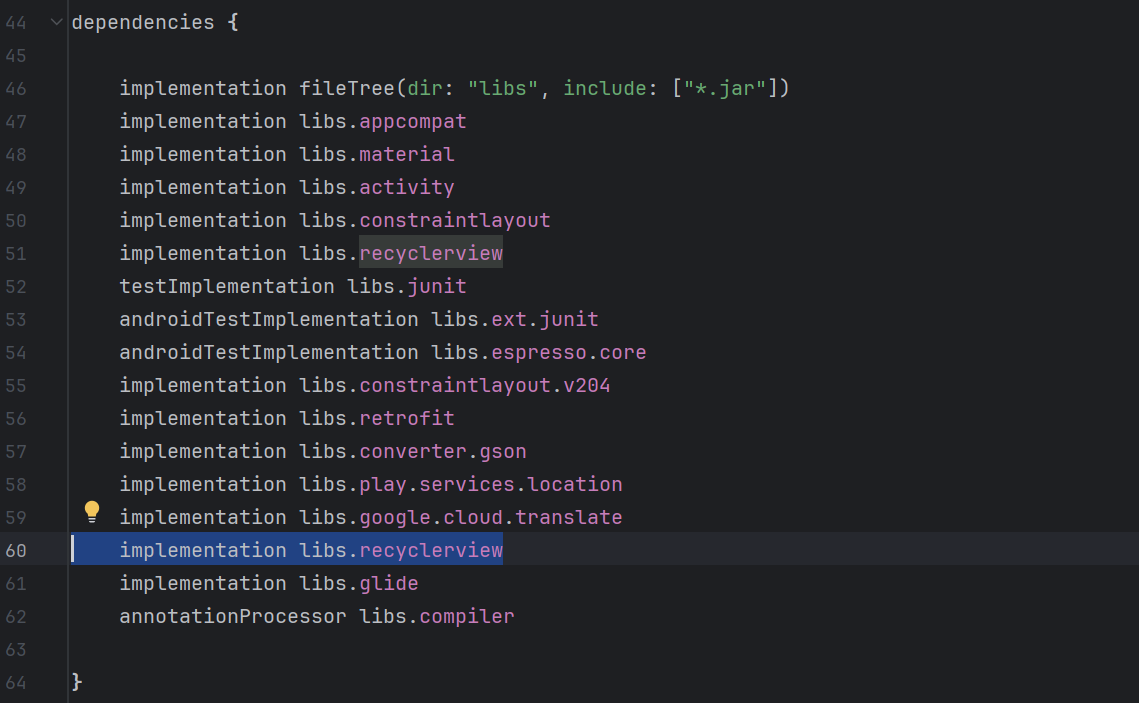
7. 在5.创建的类中重写ViewHolder方法

8. 在页面文件中，将数据传递给4.中的Adapter类，并初始化该类

9. 在页面文件中，设置RecyclerView的布局模式，并将RecyclerView的Adapter设置为8.的实例。

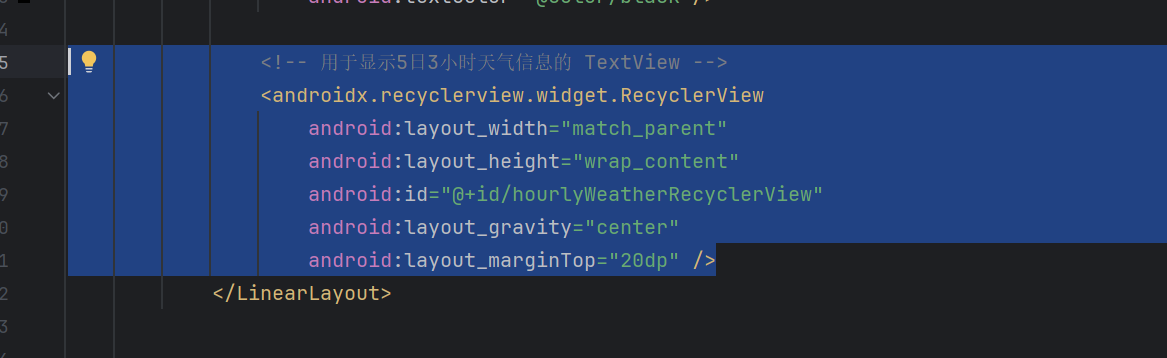
我的实现过程如下：

1. 在build.gradle文件中添加依赖



在dependencies 中，将recyclerview实现。

2. 在布局文件中添加RecyclerView布局



3. 创建RecyclerView里每个元素的布局文件

在这一步中，我将元素的布局文件放在item\_hourly\_weather.xml中，如下：

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 android:padding="10dp">  
  
 <TextView  
 android:id="@+id/dateTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Date"  
 android:textStyle="bold"  
 android:textSize="20sp"  
 android:layout\_marginLeft="20dp"  
 android:layout\_weight="1"  
 android:textColor="@color/black"/>  
  
 <TextView  
 android:id="@+id/timeTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Time"  
 android:textSize="20sp"  
 android:layout\_weight="1"  
 android:textColor="@color/black"/>  
  
 <ImageView  
 android:id="@+id/weatherIconImageView"  
 android:layout\_width="30dp"  
 android:layout\_height="30dp"  
 android:layout\_weight="1"/>  
  
 <TextView  
 android:id="@+id/temperatureTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Temp"  
 android:textStyle="bold"  
 android:textSize="20sp"  
 android:layout\_weight="1"  
 android:textColor="@color/black"/>  
</LinearLayout>

这几个组件通过横向的LinerLayout布局，在最左边显示日期，第二个显示时间信息，然后是天气状况图标，最后是温度。

4. 创建一个继承自RecyclerView.Adapter的类来管理RecyclerView。

public class HourlyWeatherAdapter extends RecyclerView.Adapter<HourlyWeatherAdapter.ViewHolder> {  
 private List<WeatherForecastResponse.WeatherForecastItem> forecastList;  
  
 public HourlyWeatherAdapter(List<WeatherForecastResponse.WeatherForecastItem> forecastList) {  
 this.forecastList = forecastList;  
 }  
  
 @NonNull  
 @Override  
 public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
 View view = LayoutInflater.*from*(parent.getContext()).inflate(R.layout.*item\_hourly\_weather*, parent, false);  
 return new ViewHolder(view);  
 }  
  
 @SuppressLint("SetTextI18n")  
 @Override  
 public void onBindViewHolder(@NonNull ViewHolder holder, int position) {  
 WeatherForecastResponse.WeatherForecastItem item = forecastList.get(position);  
 holder.dateTextView.setText(getDateFromDateTime(item.getDateText()));  
 holder.timeTextView.setText(getTimeFromDateTimeText(item.getDateText()));  
 String icon = item.getWeather().get(0).getIcon();  
 Utility.*loadWeatherIcon*(icon,holder.weatherIconImageView);  
 int temp = (int)item.getMain().getTempMin();  
 holder.temperatureTextView.setText(temp+"°C");  
 }  
  
 @Override  
 public int getItemCount() {  
 return forecastList.size();  
 }  
  
 private String getTimeFromDateTimeText(String dateTimeText) {  
 // 这里根据你的日期时间格式提取时间部分  
 return dateTimeText.substring(11, 16);  
 }  
  
 private String getDateFromDateTime(String dateTime){  
 return dateTime.substring(5,10);  
 }  
   
}

5. 在4.步骤中的类中创建一个继承了RecyclerView.ViewHolder类的内部类，用来动态绑定布局容器。

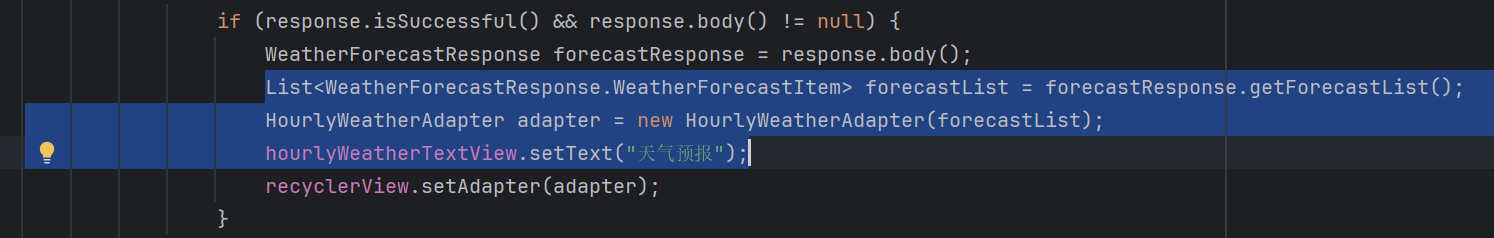
// 用来动态绑定组件  
public static class ViewHolder extends RecyclerView.ViewHolder {  
 public TextView dateTextView;  
 public TextView timeTextView;  
 public ImageView weatherIconImageView;  
 public TextView temperatureTextView;  
  
 public ViewHolder(View view) {  
 super(view);  
 dateTextView = view.findViewById(R.id.*dateTextView*);  
 timeTextView = view.findViewById(R.id.*timeTextView*);  
 weatherIconImageView = view.findViewById(R.id.*weatherIconImageView*);  
 temperatureTextView = view.findViewById(R.id.*temperatureTextView*);  
 }  
}

6. 重写4.创建的类中的一些必要的函数

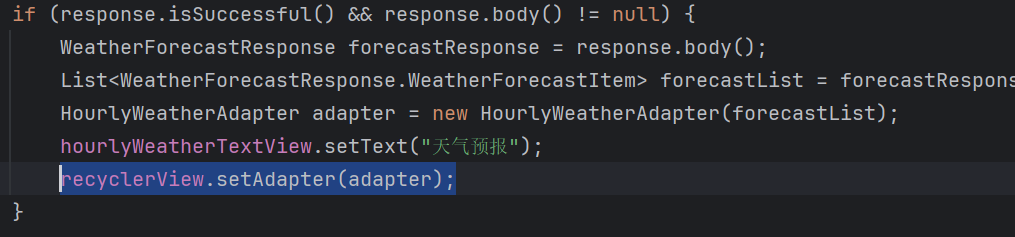
7. 在5.创建的类中重写ViewHolder方法

8. 在页面文件中，将数据传递给4.中的Adapter类，并初始化该类

我将获取的数据存入WeatherForecastResponse里面，然后通过获取里面的list信息，来得到预测的关键信息点。



9. 在页面文件中，设置RecyclerView的布局模式，并将RecyclerView的Adapter设置为8.的实例。



# 5 界面展示

## 5.1 本地天气页面





## 5.2 搜索页面



