大连海事大学2019-2020（2）学期 序号：

移动应用开发课程实践报告

项目名称：划水酱的天气预报

项目组名称：四个划水酱

项目组成员：

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 任务 | 成绩 |
| 韩雪峰 | 2220171338 |  |  |
| 严志强 | 2220171437 |  |  |
| 龙平付 | 2220173683 |  |  |
| 宋德林 | 2220172630 |  |  |

项目组成员完成工作量：

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 主要完成工作 | 占总工作量比例 |
| 韩雪峰 | 2220171338 |  |  |
| 严志强 | 2220171437 |  |  |
| 龙平付 | 2220173683 |  |  |
| 宋德林 | 2220172630 |  |  |

（提示：此页插入目录，阅读提示后删除）

（提示：以下开始报告正文）

格式要求：

中文：宋体，小四号，行距为1.5倍，段前段后0行，首行缩进2字符。

英文及数字：Times new roman，小四号，行距为1.5倍，段前段后0行，首行缩进2字符。两端对齐。

表格：三线格，字体五号，中文宋体，西文Times new roman，段前后0行，首行不缩进，1.5倍行距。按章节给表格编号并命名，表名在表上方。

图：必须用powderdesigner绘制并导出后插入到文档中，按章节给图编号，并命名。图名在图下方。

以上提示，在完成文档后，删除，并调整格式。

-----------------------------------------------------------------------------

1 项目背景

本次实验为开发一款基于Android平台的天气预报应用，让用户可以在智能手机上随时通过移动网络查询互联网所提供的天气信息。可以让Android用户能够在第一时间内获取最新的天气信息，便于提前预防，方便出行。随着移动互联网的兴起及Android机的普及, 智能手机已经成为人们日常生活中必比可少的工具。尤其国产Android手机的发展快速迅猛, 相比于苹果手机有价格上的优势, 故市场潜力巨大。目前, 手机的功能已经不仅仅是打电话发短信, 更多的是为我们的生活生产提供便利条件。以往人们不太方便获取的信息, 通过手机上网可以实时获取。比如以往人们获取天气预报的方式多为收看电视节目或收听广播, 而智能手机的出现可以通过网络更快更方便地获取天气信息。基于以上原因, 本项目制作了一款界面简洁, 操作简单的Android端天气预报应用。

2 项目设计

2.1需求分析及可行性分析

天气APP需要具备以下功能：

1.可以罗列岀全国所有的省、市、县；

2.可以查看并搜索全球任意城市的天气信息；

3.可以自由地切换城市，去查看其他城市的天气；

4.提供手动更新以及后台自动更新天气的功能；

5.登录注册功能。

6.预报当前天内两小时后的天气

7.查看AQI和PM2.5等空气质量，给出相应的生活贴士

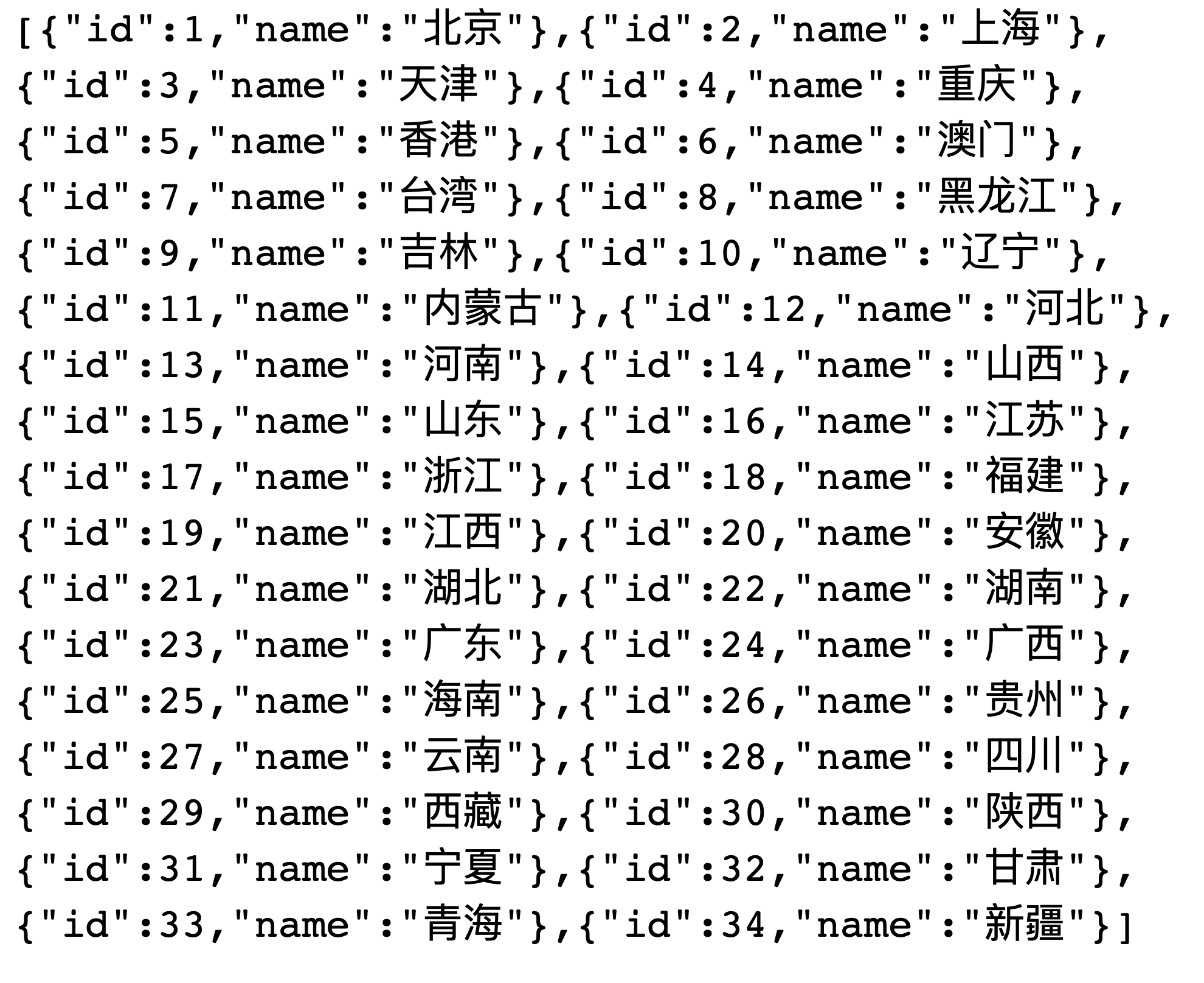
虽然看上去只有6个主要的功能点，但如果想要全部实现这些功能却需要用到UI、网络、 数据存储、服务等技术，因此还是非常考验我们的综合应用能力的。

首先需要考虑的一个问题就是，如何才能得到全国省市县的数据信息，以及如何才能获取到每个城市的天气信息。比较遗憾的是，现在网上免费的天气预报接口已经越来越少，很多之前可以使用的接口都关闭掉了。因此，这次我们特意去找了一些稳定的天气预报服务，比如彩云天气以及和风天气。这两个天气预报服务虽然都是收费的，但它们每天都提供一定次数的免费天气预报请求。其中彩云天气的数据更加实时和专业，可以将天气预报精确到分钟级，每天提供1000次免费请求；和风天气的数据相对简单一些，比较适合我们的实验，每天提供3000次免费请求。简单起见，我们就使用和风天气来作为天气预报的数据来源，每天3000次的免费请求对于此次实验而言已经是相当充足了。

解决了天气数据的问题，接下来还需要解决全国省市县数据的问题。为了方便实验，我们访问了别人架设的服务器用于提供全国所有省市县的数据信息，下面是这些接口的具体用法：比如要想罗列出中国所有的省份，只需访问如下地址：

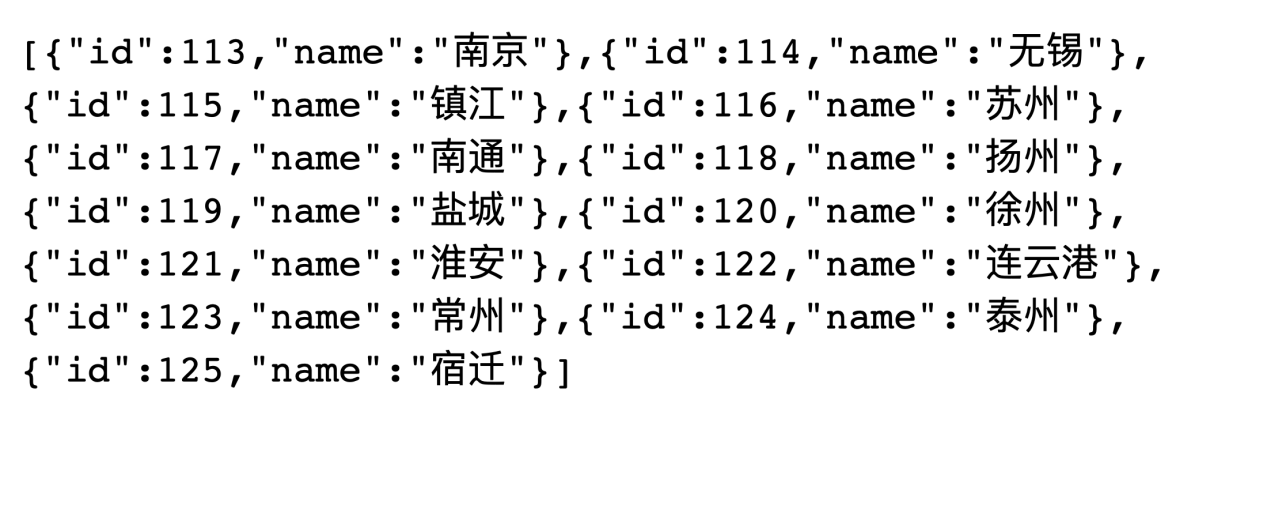
<http://guolin.tech/api/china>

服务器会返回我们一段JSON格式的数据，其中包含了中国所有的省份名称以及省份id,如下所示：



这数组中的每一个元素都代表着一个省份，其中，北京的id 是1,上海的id是2。如果想要知道某个省内有哪些城市，比如江苏的id 是16,访问如下地址即可：http://guolin.tech/api/china/16

即只需要将省份id添加到url地址的最后面就可以了，现在服务器返回的数据如下:



通过这种方式，就能把全国所有的省、市、县都罗列出来了。那么解决了省市县数据的获取，我们还需解决怎样才能查看到具体的天气信息，这就必须要用到每个地区对应的天气id 了。 每个县或区都会有一个weather\_id,拿着这个id再去访问和风天气的接口，就能够获取到该地区具体的天气信息了。

关于和风天气的接口该如何使用。首先注册一个自己的账号，注册地址是https://id.heweather.com，注册好了之后使用这个账号登录，就能看到自己的API Key,以及每天剩余的访问次数了。有了 API Key,再配合刚才的weather\_id,我们就能获取到任意城市的天气信息了。比如说 苏州的weather\_id是CN101190401,那么访问如下接口即可查看苏州的天气信息：

http://guolin.tech/api/weather?cityid=CN101190401&key=bc0418b57b2d4918819d3974acl2 85d9

其中，cityid部分填入的就是待查看城市的weather\_id, key部分填入的就是我们申请到的 API Keyo这样，服务器就会把苏州详细的天气信息以JSON格式返回给我们了。

2.2概要设计

基于Android平台进行的天气预报移动应用开发。

**2.3.1可以罗列岀全国所有的省、市、县；**

首先按照全国各地区省、市、县的分布情况，从网络接口处获取及时精准的省、市、县分布数据。按照省份、市、县（区）的结构来准确的对自己的地理位置进行精准的定位。按照分层的方法，将获取的设计数据库。将获取的数据存入省份、市、县的表中。然后建立对应的数据联系，并能够在对应的层次结构下遍历所有的省，在省份关联对应市的数据，市关联县区的数据。按照省、市、县优先级从高到低的方法，先选择省级的数据，依次选定市、县数据，层层选择，进行全国省、市、县的全部罗列。

**2.3.2**可以查看全国任意城市的天气信息；

首先把全国各地的天气情况进行一个总括，在较成熟的天气预测技术，对多个天气预测分析来源进行了比较和检验，从地域到时空应对展开实地预测检验，选定准确预报天气，得到了该地区的天气的基本状态：时间、温度、空气状况。然后在调查基础上选用了基本的信息后，便能够及时的查看对应的天气情况。

**2.3.3**可以自由地切换城市，去查看其他城市的天气；

在完成固定账号和对应位置选定和天气信息的更新后，叠加功能，使得可以在获取固定城市的天气情况后，能够更合理地展开对特定地区，在对省、市、县数据的遍历后得到对应的数据信息。

**2.3.4**.提供手动更新以及后台自动更新天气的功能；

对获取的天气数据进行更新，以能够及时的获取最想的天气情况，使用了数据的一部分，还可以为了数据的实时性设置自动的数据更新与获取进行一部分的数据获取。在后台自动更新时需要我们划分间隔，及时做好实时的天气的获取。

**2.3.5**登录注册功能。

为了实现对个人用户的优化体验，我们将实现注册与登陆功能，并可以在登陆后进行天气情况的查看和获取特定地区的天气情况以及出行的建议。在注册登陆后，可以同步AQI的和PM2.5等天气中的另一些状况。

**2.3.6**报当前两小时后的天气。

对天气情况进行详细的预测，能够获得分时段的天气情况，从系统内返回。对当下天气的数据能够及时从数据源获取。获得相应的天气情况，并进一步显示未来时段内的一个天气情况变化，更迅速，精准。

**2.3.7**查看AQI(天气质量指数)和PM2.5等空气质量内容，给出相应生活贴士。

在进行相关性设计，需要及时的得到更合适的天气状况，便于对个人的日常活动与工作有所依据。进一步的实现AQI，则是与空气质量与PM2.5的数据平台进行合理的观测并获取准确的数据，提供一些天气建议。

界面UI：需要整体式布局，分层显示，能够在登陆界面跳转，主界面设计显示主要天气状况。界面优化，在合理的分区后，对主界面背景进行优化，装饰优美的图片。

2.3详细设计

详细设计阶段主要对概要设计的内容进行代码逻辑设计等。

**2.3.1 创建数据库和表**

第一阶段我们要做的就是创建数据库和表，这样从服务器获取到的数据才能够存储到本地。这里我们准备使用LitePal来管理天气APP的数据库。

需要对项目所需的各种依赖库进行声明：

准备声明的4个库：LitePal用于对数据库进行操作，OkHttp用于进行网络请求，GSON用于解析JSON数据，Glide用于加载和展示图片。天气APP将会对这几个库进行综合运用。

设计数据库的表结构：

准备建立3张表：province. city、county,分别用于存放省、市、 县的数据信息。对应到实体类中的话，就建立Province. City、County这3个类。

实体类的主要内容就是声明了一些需要的字段，并生成相应的getter 和setter方法就可以了。

**2.3.2 遍历全国省市县数据**

全国所有省市县的数据都是从服务器端获取到的，因此这里和服务器的交互 是必不可少的，所以我们在util包下先增加一个Httputil类，目的是发起一条HTTP请求只需要调用相应方法，传入请求地址，并注册一个回调来处理服务器响应就可以了。

另外，由于服务器返回的省市县数据都是JSON格式的，所以我们需要提供一个工具类来解析和处理这种数据。在util包下新建一个Utility类，在这个类中，我们提供了 handleProvincesResponse()、handleCitiesResponse()、 handleCountiesResponseO这3个方法，分别用于解析和处理服务器返回的省级、市级和县级数据。处理的方式都是类似的，先使用JSONArray和JSONObject将数据解析出来，然后组装成实体类对象，再调用save()方法将数据存储到数据库当中。由于这里的JSON数据结构比较简单，我们就不使用GSON来进行解析了。

现在可以开始写界面了。由于遍历全国省市县的功能我们在后面还会复用，因此就不写在活动里面了，而是写在碎片里面，这样需要复用的时候直接在布局里面引用碎片就可以了。

对于布局文件中的内容，我们的方案是：先定义一个头布局来作为标题栏，设置布局高度，设置背景色。然后在头布局中放置了一个TextView用于显示标 题内容，放置了一个Button用于执行返回操作，我们已经提前准备好了一张.png图片用于作为按钮的背景图。这里之所以要自己定义标题栏，是因为碎片中最好不要直接使用 ActionBar或Toolbar,不然在复用的时候可能会出现一些不想看到的效果。

接下来在头布局的下面定义了一个ListView,省市县的数据就将显示在这里。之所以这次使用了 ListView,是因为它会自动给每个子项之间添加一条分隔线，而如果使用RecyclerView想实 现同样的功能则会比较麻烦，这里我们总是选择最优的实现方案。

接下来，我们需要编写用于遍历省市县数据的碎片了。新建Choose- AreaFragment继承自Fragment：

这个类里的代码逻辑：在onCreateViewO方法中先是获取到了一些控件的实例，然后去初始化了 ArrayAdapter,并将它设置为List View的适配器。接着在0nActivityCreated()方法中给ListView和Button设置了点击事件，到这里我 们的初始化工作就算是完成了。

在0nActivityCreated()方法的最后，调用了 queryProvinces ()方法，也就是从这里开始加载省级数据的。queryProvinces()方法中首先会将头布局的标题设置成中国，将返回按钮隐藏起来，因为省级列表已经不能再返回了。然后调用LitePal的查询接口来从数据库中读取省级数据，如果读取到了就直接将数据显示到界面上，如果没有读取到接口组装出一个请求地址，然后调用queryFromServerO法来从服务器上查询数据。

queryFromServerO方法中会调用HttpUtil的sendOkHttpRequest ()方法来向服务器发送 请求，响应的数据会回调到onResponse()方法中，然后我们在这里去调用Utility的 handleProvincesResponse()方法来解析和处理服务器返回的数据，并存储到数据库中。接下来，在解析和处理完数据之后，我们再次调用了 queryProvinces()方法来重新加载省级数据，由于queryProvinces()方法牵扯到了 UI操作，因此必须要在主线程中调用，这里借助了runOnUiThreadO方法来实现从子线程切换到主线程。现在数据库中已经存在了数据, 因此调用queryProvinces ()就会直接将数据显示到界面上了。

当用户点击了某个省的时候会进入到ListView的onItemClick()方法中，这个时候会根据当前的级别来判断是去调用queryCities()方法还是queryCounties()方法，queryCities ()方法是去查询市级数据，而queryCounties()方法是去查询县级数据，这两个方法内部的流程和 queryProvinces()方法基本相同。另外还有一点需要注意，在返回按钮的点击事件里，会对当前ListView的列表级别进行判断。 如果当前是县级列表，那么就返回到市级列表，如果当前是市级列表，那么就返回到省级表列表。 当返回到省级列表时，返回按钮会自动隐藏，从而也就不需要再做进一步的处理了。

这样我们就把遍历全国省市县的功能完成了，可是碎片是不能直接显示在界面上的，因此还需要把它添加到活动里才行。

另外，我们刚才在碎片的布局里面已经自定义了一个标题栏，因此就不再需要原生的 ActionBar 了。

最后还需声明程序所需要的权限。

**2.3.3 显示天气信息**

在第三阶段中，我们就要开始去查询天气，并且把天气信息显示出来了。由于和风天气返回的JSON数据结构非常复杂，如果还使用JSONObject来解析就会很麻烦，这里我们就准备借助 GSON来对天气信息进行解析了。

**2.3.3.1定义GSON实体类**

GSON的用法很简单，解析数据只需要一行代码就能完成了，但前提是要先将数据对应的实体类创建好。由于和风天气返回的数据内容非常多，这里我们不可能将所有的内容都利用起来， 因此筛选了一些比较重要的数据来进行解析。

设计返回数据的大致格式：

{

"HeWeather":[

{

"status": "ok",

"basic": {},

"aqi": {},

"now": {},

"suggestion": {}, "dailyforecast":[]

}

]

}

其中，basic、aqi、now、suggestion和daily\_forecast的内部又都会有具体的内容，那么我们就可以将这5个部分定义成5个实体类。

basic中具体内容如下所示：

“basic”:{

“city” ：“ 苏州”，

“id”:"CN101190401”，

“update”:{

“loc”：“2016M8M8 21:58H”

}

}

其中，city表示城市名，id表示城市对应的天气id, update中的loc表示天气的更新时间。我们按照此结构就可以在gson包下建立一个Basic类。

将Basic类定义好后,其余的几个实体类也是类似的，我们使用同样的方式来定义就可以了。

把basic, aqi、now、suggestion和daily\_forecast对应的实体类全部都创建好后，接下来还需要再创建一个总的实例类来引用刚刚创建的各个实体类。在gson包下新建一个Weather类。

在 Weather 类中，我们对 Basic、AQI、Now、Suggestion 和 Forecast 类进行了引用。其中，由于daily forecast中包含的是一个数组，因此这里使用了 List集合来引用Forecast类。 另外，返回的天气数据中还会包含一项status数据，成功返回ok,失败则会返回具体的原因，那么这里也需要添加一个对应的status字段。

**2.3.3.2 编写天气界面**

创建一个用于显示天气信息的活动。由于所有的天气信息都将在同一个界面上显示，因此activity weather.xml会是一个很长的布局文件。那么为了让里面的代码不至于混乱不堪，我们将界面的不同部分写在不同的布局文件里面，再通过引入布局的方式集成到activity, weather.xml中，这样整个布局文件就会显得非常工整。

建一个title.xml作为头布局，在这段代码中，头布局中放置了两个TextView, 一个居中显示城市名，一个居右显示更新时间。

然后建一个now.xml作为当前天气信息的布局，当前天气信息的布局中也是放置了两个TextView, 一个用于显示当前气温，一个用于显示天气概况。

然后建forecast.xml作为未来几天天气信息的布局，这里最外层准备使用LinearLayout定义了一个半透明的背景，然后使用TextView定义了一个标题，接着使用一个LinearLayout定义了一个用于显示未来几天天气信息的布局。不过这个布局中并不打算放入任何内容，因为这是要根据服务器返回的数据在代码中动态添加的。

为此，我们还需要再定义一个未来天气信息的子项布局，创建forecast item.xml文件，子项布局中放置了 4个TextView, 一个用于显示天气预报日期，一个用于显示天气概况，另外两个分别用于显示当天的最高温度和最低温度。

然后建aqi.xml作为空气质量信息的布局，这个布局中的代码中使用 LinearLayout定义了一个半透明的背景，然后使用TextView定义了一个标题。接下来，使用 LinearLayout和RelativeLayout嵌套的方式实现了一个左右平分并且居中对齐的布局，分别用于显示AQI指数和PM2.5指数。

然后新建suggestion.xml作为生活建议信息的布局，先定义了一个半透明的背景和一个标题，然后下面使用了 3个TextView分别 用于显示舒适度、洗车指数和运动建议的相关数据。

当我们把天气界面上每个部分的布局文件都编写好后，接下来的就是将它们引入到 activity weather.xml 当中。首先最外层布局使用了一个FrameLayout,并将它的背景色设置成colorPrimary0 然后在FrameLayout中嵌套了一个ScrollView,这是因为天气界面中的内容比较多，使用 ScrollView可以允许我们通过滚动的方式查看屏幕以外的内容。

**2.3.3.3 将天气显示到界面上**

首先在Utility类中添加一个用于解析天气JSON数据的方法，然后由于之前已经按照上面的数据格式定义过相应的GSON实体类，因此只需要通过调用fromJsonO方法就能直接将JSON数据转换成Weather对象了。

接下来的工作是我们如何在活动中去请求天气数据，以及将数据展示到界面上：

在requestWeatherO方法中先是使用了参数中传入的天气id和我们之前申请好的API Key拼装出一个接口地址，接着调用Httplltil. sendOkHttpRequest ()方法来向该地址发出请求，服务器会将相应城市的天气信息以JSON格式返回。然后我们在onResponseO回调中先调用Utility. handleWeatherResponseO方法将返回的JSON数据转换成Weather对象，再将当前线程切换到主线程。然后进行判断，如果服务器返回的status状态是ok,就说明请求天气成功了，此时将返回的数据缓存到SharedPreferences当中，并调用showWeatherlnfo()方法来进行内容显示。showWeatherlnfoO方法中的逻辑就比较简单了，其实就是从Weather对象中获取数据，然后显示到相应的控件上。注意在未来几天天气预报的部分我们使用了一个for循环来处理每天的天气信息，在循环中动态加载forecast\_item.xml布局并设置相应的数据，然后添加到父布局当中。设置完了所有数据之后，记得要将ScrollView重新变成可见。

这样我们就将首次进入WeatherActivity时的逻辑全部梳理完了，那么当下一次再进入 WeatherActivity时，由于缓存已经存在了，因此会直接解析并显示天气数据，而不会再次发起网络请求了。

处理完了 WeatherActivity中的逻辑，接下来我们要做的，就是如何从省市县列表界面跳转到天气界面了。

定义onCreate()方法的一开始先从SharedPreferences文件中读取缓存数据，如果不为null就说明之前已经请求过天气数据了，那么就没必要让用户再次选择城市，而是直接跳转到WeatherActivity即可。

**2.3.3.4 获取必应每日一图**

每日一图的图片都是由必应精挑细选出来的，并且每天都会变化，如果我们使用它们来作为天气界面的背景图，不仅可以让界面变得更加美观，而且解决了界面一成不变、过于单调的问题。

为此访问别人架设的专门的一个获取必应每日一图的接口 ：

这里我们在FrameLayout中添加了一个Image View,并且将它的宽和高都设置成match\_ parento由于FrameLayout默认情况下会将控件都放置在左上角，因此ScrollView会完全覆盖住 Image View,从而Image View也就成为背景图片了。

接着修改WeatherActivity中的代码。

首先在onCreate()方法中获取了新增控件ImageView的实例，然后尝试从 SharedPreferences中读取缓存的背景图片。如果有缓存的话就直接使用Glide来加载这张图片，如果没有的话就调用loadBingPic()方法去请求今日的必应背景图。

loadBingPic()方法中的逻辑：先是调用了 HttpUtil.sendOkHttpRequest() 方法获取到必应背景图的链接，然后将这个链接缓存到SharedPreferences当中，再将当前线程切换到主线程，最后使用Glide来加载这张图片就可以了。另外需要注意，在requestWeatherO 方法的最后也需要调用一下loadBingPic()方法，这样在每次请求天气信息的时候同时也会刷新背景图片。

修改WeatherActivity 中的代码，将背景图和状态栏融合到一起。由于这个功能是Android 5.0及以上的系统才支持的，因此我们先在代码中做了一个系统版本号的判断，只有当版本号大于或等于21,也就是5.0及以上系统时才会执行后面的代码。

接着我们调用getWindow() ,getDecorView()方法拿到当前活动的DecorView,再调用它 的setSystemUiVisibility()方法来改变系统 UI的显示，这里传入View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLS和View. SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE 就表示活动的布局会显示在状态栏上面，最后调用一下setStatusBarColor()方法将状态栏设置成透明色。

这些代码就可以实现让背景图和状态栏融合到一起的效果了。不过，发现还是有些问题，天气界面的头布局几乎和系统状态栏紧贴到一起了。

这是由于系统状态栏已经成为我们布局的一部分，因此没有单独为它留出空间。我们借助android:fitsSystemWindov/s属性就可以了。

**2.3.4 手动更新天气和切换城市**

经过以上阶段的开发，现在天气APP的主体功能已经有了。因此，在第下一步中我们要加入切换城市的功能，并且为了能够实时获取到最新的天气，我们还会加入手动更新天气的功能。

**2.3.4.1 手动更新天气**

实现手动更新天气的功能。由于我们在上一节中对天气信息进行了缓存，目前每次展示的都是缓存中的数据，因此现在非常需要一种方式能够让用户手动更新天气信息。

这里我准备采用下拉刷新的方式触发更新事件及加入更新天气的处理逻辑。

首先在onCreate()方法中获取到了 SwipeRefreshLayout的实例，然后调用setColorSchemeResources()方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的colorPrimary作为进度条的颜色了。接着定义了一个weatherld变量，用于记录城市的天气 id,然后调用setOnRef reshListener()方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候，就会回调这个监听器的onRefresh()方法，我们在这里去调用requestWeather()方法请求天气信息就可以了。

另外，当请求结束后，还需要调用SwipeRefreshLayout的setRef reshing ()方法并传入false,用于表示刷新事件结束，并隐藏刷新进度条。

**2.3.4.2 切换城市**

要切换城市，那么就需要遍历全国省市县的数据，而这个功能我们早在前面就已经完成了，并且当时考虑为了方便后面的复用，特意选择了在碎片当中实现。因此，我们其实只需要在天气界面的布局中引入这个碎片，就可以快速集成切换城市功能了。

虽说实现原理简单，但是我们也不可能让引入的碎片把天气界面遮挡住，将碎片放入到滑动菜单中，正常情况下它不占据主界面的任何空间，想要切换城市的时候只需要通过滑动的方式将菜单显示出来就可以了。

首先按照Material Design，我们需要在头布局中加入一个切换城市的按钮，不然的话用户可能根本就不知道屏幕的左侧边缘是可以拖动的。

这里添加了一个Button作为切换城市的按钮，并且让它居左显示。提前准备好了一张图片来作为按钮的背景图。

接下来需要在WeatherActivity中加入滑动菜单的逻辑处理：

首先在onCreate()方法中获取到新增的DrawerLayout和Button的实例，然后在 Button的点击事件中调用DrawerLayout的openDrawer()方法来打开滑动菜单就可以了。

这仅仅是打开了滑动菜单，我们还需要处理切换城市后的逻辑。这就必须要在ChooseAreaFragment中进行了，因为之前选中了某个城市后是跳转到WeatherActivity的,而现在由于我们本来就是在WeatherActivity当中的，因此并不需要跳转, 只是去请求新选择城市的天气信息就可以了。

这里还需要根据ChooseAreaFragment的不同状态来进行不同的逻辑处理。

**2.3.5 后台自动更新天气**

为了要让天气APP更加智能，在这阶段我们准备加入后台自动更新天气的功能，这样就可以尽可能地保证用户每次打开软件时看到的都是最新的天气信息。

要想实现上述功能，就需要创建一个长期在后台运行的定时任务。

在onStartCommandO方法中先是调用了 updateWeather()方法来更新天气， 然后调用了 updateBingPicO方法来更新背景图片。这里我们将更新后的数据直接存储到 SharedPreferences文件中就可以了，因为打开WeatherActivity的时候都会优先从SharedPreferences 缓存中读取数据。

之后就是创建定时任务了，将时间间隔设置为8小时，8小时后AutoUpdateReceiver的onStartCommand ()方法就会重新执行， 这样也就实现后台定时更新的功能了。

不过，我们还需要在代码某处去激活AutoUpdateService这个服务才行。

这里在showWeather()方法的最后加入启动AutoUpdateService这个服务的代码， 这样只要一旦选中了某个城市并成功更新天气之后，AutoUpdateService就会一直在后台运行，并保证每8小时更新一次天气。

**2.3.6 修改图标和名称**

将图片命名成logo.png,放入到所有以mipmap开头的目录下，然后修改AndroidManifest.xml中的代码，如下所示：

<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.coolweather.android">

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

<application

android:name="org.litepal.LitePalApplication"

android:allowBackup="true"

android: icon=,l@mipmap/'logo,>

and roid: label="(ast ring/appname" android:supportsRtl="true" and roid: theme="@sty'le/AppTheme">

</application>

</manifest>

这里将<application>标签的android:icon属性指定成@mipmap/logo就可以修改程序图标了。接下来我们还需要修改一下程序的名称，打开res/values/string.xml文件，其中app name 对应的就是程序名称，将它修改即可，如下所示：

<resources>

<string name="app\_name”>天气v/string〉

</resources>

**2.3.7 用sqllite实现注册登录**

3 代码实现

详细设计阶段主要对概要设计的内容进行代码逻辑设计等。

**2.3.1 创建数据库和表**

第一阶段我们要做的就是创建好数据库和表，这样从服务器获取到的数据才能够存储到本地。这里我们准备使用LitePal来管理天气APP的数据库。

首先需要将项目所需的各种依赖库进行声明，编辑app/build.gradle文件，在dependencies闭 包中添加如下内容：

dependencies {

compile fileTree(dir: 'libs', include: [jar'])

compile 1 com.android.support:appcompat-v7:24.2.1'

testCompile 1junit:junit:4.12'

compile 'org.litepal.android:core:1.3.2\*

compile 'com・squareup.okhttp3:okhttp:3.4.1\*

compile 'com.google.code・gson:gson:2.7'

compile 'com.github.bumptech.glide:glide:3.7.0'

}

这里声明的4个库，LitePal用于对数据库进行操作，OkHttp用于进行网络请求，GSON用于解析JSON数据，Glide用于加载和展示图片。天气APP将会对这几个库进行综合运用，这里直接一次性将它们都添加进来。

设计数据库的表结构：

准备建立3张表：province. city、county,分别用于存放省、市、 县的数据信息。对应到实体类中的话，就建立Province. City、County这3个类。

新建一个Province类：

public class Province extends DataSupport {

private int id;

private String provinceName;

private int provinceCode;

return id;

public void setld(int id) { this.id = id;

}

public String getProvinceName() {

return provinceName;

}

public void setProvinceName(String provinceName) {

this.provinceName = provinceName;

}

public int getProvinceCode() {

return provinceCode;

}

public void setProvinceCode(int provinceCode) {

this.provinceCode = provinceCode;

}

}

其中，id是每个实体类中都应该有的字段，provinceName记录省的名字，provinceCode 记录省的代号。另外，LitePal中的每一个实体类都是必须要继承自DataSupport类的。

接着新建一个City类，代码如下所示:

public class City extends DataSupport {

private int id;

private String cityName;

private int cityCode;

private int provinceld;

public int getld() {

return id;

}

public void setld(int id) {

this.id = id;

}

public String getCityName() {

return cityName;

}

public void setCityName(String cityName) {

this.cityName = cityName;

public int getCityCode() { return cityCode;

public void setCityCode(int cityCode) { this.cityCode = cityCode;

}

public int getProvinceId() {

return provinceld;

}

public void setProvinceId(int provinceld) { this.provinceld = provinceld;

}

}

其中，cityName记录市的名字，cityCode记录市的代号，provinceld记录当前市所属省 的id值。

新建一个County类，代码如下所示：

public class County extends DataSupport {

private int id;

private String countyName;

private String weatherld;

private int cityld;

public int getld() {

return id;

public void setld(int id) { this.id = id;

public String getCountyName() { return countyName;

}

public void setCountyName(St ring countyName) { this.countyName = countyName;

public String getWeatherld() { return weatherld;

public void setWeathe rid(St ring weatherld) { this.weatherld = weatherld;

}

public int getCityId() (

return cityld;

}

public void setCityId(int cityld) {

this.cityld = cityld;

}

}

其中，countyName记录县的名字，weatherld记录县所对应的天气id, cityld记录当前 县所属市的id值。

实体类的主要内容就是声明了一些需要的字段，并生成相应的getter 和setter方法就可以了。

接下来需要配置litepal.xml文件。右击app/src/main目录—>New—>Directory,创建一个assets 目录，然后在assets目录下再新建一个litepal.xml文件，接着编辑1itepal.xml文件中的内容，如下所示：

<litepal>

<dbname value="cool\_weather" />

<version value="l" />

<list>

<mapping class="com.coolweather.android.db.Province" />

<mapping class="com.coolweather.android.db.City" />

<mapping class="com.coolweather.android.db.County" />

</list>

</litepal>

这里将数据库名指定成cool weather,数据库版本指定成1,并将Province. City和 County这3个实体类添加到映射列表当中。

最后还需要再配置一下LitePalApplication,修改AndroidManifest.xm1中的代码，如下所示：

<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.coolweather.android">

<application

android: name=“org.litepal.LitePalApplication”

android:allowBackup="true"

android: icon="(amipmap/ic\_launcher"

android:label="(astring/app name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

</application>

</manifest>

这样就将所有的配置都完成了，数据库和表会在首次执行任意数据库操作的时候自动创建。

**2.3.2 遍历全国省市县数据**

全国所有省市县的数据都是从服务器端获取到的，因此这里和服务器的交互 是必不可少的，所以我们在util包下先增加一个Httputil类，代码如下所示：

public class HttpUtil {

public static void sendOkHttpRequest(String address, okhttp3.Callback callback) ( OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder().url(address).build(); client.newCall(request).enqueue(callback);

}

}

现在发起一条HTTP请求只需要调用sendOkHttpRequest ()方法，传入请求地址，并注册一个回调来处理服务器响应就可以了。

另外，由于服务器返回的省市县数据都是JSON格式的，所以我们需要提供一个工具类来解析和处理这种数据。在util包下新建一个Utility类，代码如下所示：

public class Utility {

/\*\*

\*解析和处理服务器返回的省级数据

\*/

public static boolean handleProvinceResponse(String response) ( if (!TextUtils.isEmpty(response)) {

try {

JSONArray allProvinces = new JSONArray(response);

for (int i = 0; i < allProvinces.length(); i++) (

JSONObject provinceobject = allProvinces.getJSONObject(i); Province province = new Province();

province.setProvinceName(provinceObject.getString("name")); province.setProvinceCode(provinceObject.getlnt("id")); province.save();

}

return true;

} catch (JSONException e) { e.printStackTrace();

}

}

return false;

}

/\*\*

\*解析和处理服务器返回的市级数据

\*/

public static boolean handleCityResponse(String response, int provinceld) { if (JTextUtils.isEmpty(response)) {

try (

JSONArray allCities = new JSONArray(response); for (int i = 0; i < allCities.length(); i++) { JSONObject cityObject = allCities.getJSONObject(i); City city = new City();

city.setCityName(cityObject.getString("name"));

city.setCityCode(cityObject.getInt("id"));

city.setProvinceld(provinceld);

city.save();

}

return true;

} catch (JSONException e) { e.printStackTrace();

}

}

return false;

}

/\*\*

\*解析和处理服务器返回的县级数据

\*/

public static boolean handleCountyResponse(String response, int cityld) { if (ITextUtils.isEmpty(response)) {

try {

JSONArray allCounties = new JSONArray(response);

for (int i = 0; i < allCounties.length(); i++) { JSONObject countyObject = allCounties.getJSONObject(i); County county = new County();

county.setCountyName(countyObject.getString("name")); county.setWeatherld(countyObject.getString("weatherid")); county.setCityld(cityld);

county.save();

}

return true;

} catch (JSONException e) {

e.printStackTrace();

}

}

return false;

}

}

我们提供了 handleProvincesResponse()、handleCitiesResponse()、 handleCountiesResponseO这3个方法，分别用于解析和处理服务器返回的省级、市级和县级数据。处理的方式都是类似的，先使用JSONArray和JSONObject将数据解析出来，然后组装成实体类对象，再调用save()方法将数据存储到数据库当中。由于这里的JSON数据结构比较简单，我们就不使用GSON来进行解析了。

现在可以开始写界面了。由于遍历全国省市县的功能我们在后面还会复用，因此就不写在活动里面了，而是写在碎片里面，这样需要复用的时候直接在布局里面引用碎片就可以了。

在res/layout目录中新建choose area.xml布局，代码如下所示：

<LinearLayout

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

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="#fff">

<RelativeLayout

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

and roid :backg round=,,?attr/colorPrimary,,>

<TextView

android:id="@+id/title\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" android:layout\_centerInParent="true" android:textCoTor="#fff" android:textSize="20sp"/>

<Button

android :id=,,@+id/back\_button"

android:layout\_width="25dp"

android:layout\_height="25dp"

android:layout\_marginLeft="10dp"

android:layoutalignParentLeft="true"

android:layout\_centerVertical="true" android:background="@drawable/ic\_back"/> </RelativeLayout>

<ListView

android:id="@+id/list\_view" android:layout\_width="match\_parent" android:layout\_height="matchparent"/>

</LinearLayout>

对于布局文件中的内容，我们先是定义了一个头布局来作为标题栏，将布局高度设置为 actionBar的高度，背景色设置为colorPrimary。然后在头布局中放置了一个TextView用于显示标 题内容，放置了一个Button用于执行返回操作，注意我已经提前准备好了一张ic\_back.png图片 用于作为按钮的背景图。这里之所以要自己定义标题栏，是因为碎片中最好不要直接使用 ActionBar或Toolbar,不然在复用的时候可能会出现一些不想看到的效果。

接下来在头布局的下面定义了一个ListView,省市县的数据就将显示在这里。之所以这次使用了 ListView,是因为它会自动给每个子项之间添加一条分隔线，而如果使用RecyclerView想实 现同样的功能则会比较麻烦，这里我们总是选择最优的实现方案。

接下来，我们需要编写用于遍历省市县数据的碎片了。新建Choose- AreaFragment继承自Fragment,代码如下所示：

private ProgressDialog progressDialog;

private TextView titleText;

private Button backButton;

private Listview listview;

private ArrayAdapter<String> adapter;

private List<String> dataList = new ArrayList<>();

/\*\*

\*省列表

\*/

private List<Province> provinceList;

/\*\*

\*市列表

\*/

private List<City> cityList;

/\*\*

\*县列表

\*/

private List<County> countyList;

/\*\*

\*选中的省份

\*/

private Province selectedProvince;

/\*\*

\*选中的城市

\*/

private City selectedCity;

/\*\*

\*当前选中的级别

\*/

private int currentLevel;

@0verride

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedlnstanceState) {

View view = inflater.inflate(R.layout.choosearea, container, false); titleText = (Textview) view.findViewByld(R.id.titletext); backButton = (Button) view.findViewByld(R.id.backbutton); listview = (ListView) view.findViewByld(R.id.listview);

adapter = new ArrayAdapter<>(getContext(), android.R.layout.simple\_list\_ iteml, dataList);

listview.setAdapter(adapter);

return view;

}

©Override

public void onActivityCreated(Bundle savedlnstanceState) { super.onActivityCreated(savedlnstanceState); listview.setOnItemClickListener(new AdapterView.OnltemClickListenerf) { ^Override

public void onItemClick(AdapterView<?> parentz View view, int position, long id) ( if (currentLevel == LEVEL\_PROVINCE) {

selectedProvince = provinceList.get(position); queryCities();

} else *if (currentLevel* == LEVELCITY) < selectedCity = cityList.get(position); queryCounties();

}

}

})；

backButton.setOnClickListener(new View.OnClickListener() { ^Override

public void onClick(View v) {

if (currentLevel == LEVEL\_COUNTY) { queryCities();

} else if (currentLevel == LEVEL\_CITY) { queryProvinces();

}

}

})；

queryProvinces();

\*查询全国所有的省，优先从数据库查询，如果没有查询到再去服务器上查询 \*/

private void queryProvinces() { titleText.setText(”中国”)； backButton.setVisibility(View.GONE); provinceList = DataSupport.findAll(Province.class); if (provinceList.size() > 0) { dataList.clear();

for (Province province : provinceList) { dataList.add(province.getProvinceName());

} adapter.notifyDataSetChanged(); listview.setSelection(O); currentLevel = LEVELPROVINCE;

else {

String address = "<http://guolin.tech/api/china>"; queryFromSeever(address, "province");

\*查询选中省内所有的市，优先从数据库查询，如果没有查询到再去服务器上查询

\*/

private void queryCities() {

titleText.setText(selectedProvince.getProvinceName()); backButton.setvisibility(View.VISIBLE);

cityList = DataSupport.where("provinceid = ?", St ring.valueOf(selected Province.getld())).find(City.class);

if (cityList.size() > 0) {

dataList.clear();

for (City city : cityList) {

dataList.add(city.getCityName());

}

adapter.notifyDataSetChanged(); listview.setSelection(O); currentLevel = LEVEL\_CITY;

) else { 一

int provinceCode = selectedProvince.getProvinceCode();

String address = "[http://guolin.tech/api/china/](http://guolin.tech/api/china/u)[u](http://guolin.tech/api/china/u) + provinceCode; queryFromSeever(address, "city");

\*查询选中市内所有的县，优先从数据库查询，如果没有查询到再去服务器上查询

\*/

private void queryCounties() {

titleText.setText(selectedCity.getCityName()); backButton.setvisibility(View.VISIBLE);

countyList = DataSupport.where("cityid = ?", St ring.valueOf(selectedCity. getld())).find(County.class);

if (countyList.size() > 0) { dataList.clear(); for (County county : countyList) { dataList.add(county.getCountyName());

}

adapter.notifyDataSetChangedf);

listview.setSelection(O); currentLevel = LEVEL\_COUNTY;

} else {

int provinceCode = selectedProvince.getProvinceCodef);

int cityCode = selectedCity.getCityCode();

String address = "<http://guolin.tech/api/china/>" + provinceCode + "/" + cityCode;

que ryF romSe rve r(add res s, "county");

\*根据传入的地址和类型从服务器上查询省市县数据

\*/

private void que ryF romSe rve r(St ring address, final String type) { showProgressDialog(); HttpUtil.sendOkHttpRequest(address, new Callback。 {

(aOverride

public void onResponse(Call call, Response response) throws lOException { String responseText = response.body().string(); boolean result = false; if ("province".equals(type)) {

result = Utility.handleProvinceResponse(responseText);

} else if ("city".equals(type)) ( result = Utility.handleCityResponse(responseText, selectedProvince.getId());

} else if ("county".equals(type)) { result = Utility.handleCountyResponse(responseText, selectedCity.getld());

}

if (result) {

getActivity().runOnUiThread(new Runnable() {

^Override

public void run() { closeProgressDialog(); if ("province".equals(type)) (

queryProvinces();

} else if ("city".equals(type)) { queryCities();

} else if ("county".equals(type)) { queryCounties();

}

}

})；

)

}

(QOverride

public void onFailure(Call call, lOException e) {

//通过runOnlliThread()方法回到主线程处理逻辑

getActivity(). runOnlliThread(new Runnable() {

^Override

public void run() { closeProgressDialog(); Toast.makeText(getContext(),"加载失败“，Toast.LENGTH SHORT). show();

}

})；

}

})；

}

/\*\*

\*显示进度对话框

\*/

private void showProgressDialog() (

if (progressDialog == null) { progressDialog = new ProgressDialog(getActivity()); progressDialog. setMessage("正在加载 progressDialog.setCanceledOnTouchOutside(false);

} progressDialog.show();

}

/\*\*

\*关闭进度对话框

\*/

private void closeProgressDialog() {

if (progressDialog != null) { progressDialog.dismiss();

}

}

这个类里的代码可是逻辑：在onCreateViewO 方法中先是获取到了一些控件的实例，然后去初始化了 ArrayAdapter,并将它设置为List View的 适配器。接着在0nActivityCreated()方法中给ListView和Button设置了点击事件，到这里我 们的初始化工作就算是完成了。

在0nActivityCreated()方法的最后，调用了 queryProvinces ()方法，也就是从这里开始加载省级数据的。queryProvinces()方法中首先会将头布局的标题设置成中国，将返回按钮隐藏起来，因为省级列表已经不能再返回了。然后调用LitePal的查询接口来从数据库中读取省级数据，如果读取到了就直接将数据显示到界面上，如果没有读取到接口组装出一个请求地址，然后调用queryFromServerO法来从服务器上查询数据。

queryFromServerO方法中会调用HttpUtil的sendOkHttpRequest ()方法来向服务器发送 请求，响应的数据会回调到onResponse()方法中，然后我们在这里去调用Utility的 handleProvincesResponse()方法来解析和处理服务器返回的数据，并存储到数据库中。接下来，在解析和处理完数据之后，我们再次调用了 queryProvinces()方法来重新加载省级数据，由于queryProvinces()方法牵扯到了 UI操作，因此必须要在主线程中调用，这里借助了runOnUiThreadO方法来实现从子线程切换到主线程。现在数据库中已经存在了数据, 因此调用queryProvinces ()就会直接将数据显示到界面上了。

当用户点击了某个省的时候会进入到ListView的onItemClick()方法中，这个时候会根据当前的级别来判断是去调用queryCities()方法还是queryCounties()方法，queryCities ()方法是去查询市级数据，而queryCounties()方法是去查询县级数据，这两个方法内部的流程和 queryProvinces()方法基本相同。另外还有一点需要注意，在返回按钮的点击事件里，会对当前ListView的列表级别进行判断。 如果当前是县级列表，那么就返回到市级列表，如果当前是市级列表，那么就返回到省级表列表。 当返回到省级列表时，返回按钮会自动隐藏，从而也就不需要再做进一步的处理了。

这样我们就把遍历全国省市县的功能完成了，可是碎片是不能直接显示在界面上的，因此还需要把它添加到活动里才行。修改activity\_main.xml中的代码，如下所示：

<FrameLayout

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<fragment

android:id="@+id/choose\_areafragment" android:name="com.coolweather.android.ChooseAreaFragment" android:layout\_width="match\_parent" android:layout\_height="match\_parent" />

</FrameLayout>

布局文件很简单，只是定义了一个FrameLayout,然后将ChooseAreaFragment添加进来，并让它充满整个布局。

另外，我们刚才在碎片的布局里面已经自定义了一个标题栏，因此就不再需要原生的 ActionBar 了，修改 res/values/styles.xml 中的代码，如下所示：

<resources>

<!-- Base application theme.-->

<style name="AppThemeH parent="Theme.AppCompat.Light.NoActionBar">

</style>

</resources>

还需声明程序所需要的权限。修改AndroidManifest.xml中的代码，如下 所示：

<manifest

xmlns:android="<http://schemas.android.com/apk/res/android>" package="com.coolweather.androidn>

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

**2.3.3 显示天气信息**

在第三阶段中，我们就要开始去查询天气，并且把天气信息显示出来了。由于和风天气返回的JSON数据结构非常复杂，如果还使用JSONObject来解析就会很麻烦，这里我们就准备借助 GSON来对天气信息进行解析了。

**2.3.3.1定义GSON实体类**

GSON的用法很简单，解析数据只需要一行代码就能完成了，但前提是要先将数据对应的实 体类创建好。由于和风天气返回的数据内容非常多，这里我们不可能将所有的内容都利用起来， 因此筛选了一些比较重要的数据来进行解析。

返回数据的大致格式：

{

"HeWeather":[

{

"status": "ok",

"basic": {},

"aqi": {},

"now": {},

"suggestion": {}, "dailyforecast":[]

}

]

}

其中，basic、aqi、now、suggestion和daily\_forecast的内部又都会有具体的内容， 那么我们就可以将这5个部分定义成5个实体类。

basic中具体内容如下所示：

“basic”:{

“city” ：“ 苏州”，

“id”:"CN101190401”，

“update”:{

“loc”：“2016M8M8 21:58H”

}

}

其中，city表示城市名，id表示城市对应的天气id, update中的loc表示天气的更新时间。我们按照此结构就可以在gson包下建立一个Basic类，代码如下所示：

public class Basic {

@SerializedName("city")

public String cityName;

@SerializedName("id")

public String weatherld;

public Update update;

public class Update {

@SerializedName("loc")

public String updateTime;

}

}

由于JSON中的一些字段可能不太适合直接作为Java字段来命名，因此这里使用了 @SerializedName注解的方式来让JSON字段和Java字段之间建立映射关系。

这样我们就将Basic类定义好了,其余的几个实体类也是类似的，我们使用同样的方式来定义就可以了。比如api中的具体内容如下如示：

“aqi”：{

"city":{

"aqi" :“44”,

}

}

那么,在gson包下新建一个AQI类，代码如下所示：

public class AQI {

public AQICity city;

public class AQICity {

public String aqi;

public String pm25;

}

}

now中的具体内容如下所示:

"now"：{

,'cond":{

"txt":"阵雨"

}

}

在gson包下新建一个Now类，代码如下所示:

public class Now {

@SerializedName("tmp") public String temperature;

@SerializedName("cond")

public More more;

public class More {

@SerializedName("txt") public String info;

}

suggestion中的具体内容如下所示：

"suggestion":{

"comf \*\*: (

"txt":"白天天气较热，虽然有雨，但仍然无法削弱较高气温给人们带来的暑意， 这种天气会让您感到不很舒适。”

},

"cw":{

"txt" 不宜洗车，未来24小时内有雨，如果在此期间洗车，雨水和路上的泥水 可能会再次弄脏您的爱车。"

},

"sport":{

有降水，且风力较强，推荐您在室内进行低强度运动；若坚持户外运动， 请选择避雨防风的地点。"

}

在gson包下新建一个Suggestion类，代码如下所示:

public class Suggestion {

@SerializedName("comf")

public Comfort comfort;

@SerializedName("cw")

public CarWash carWash;

public Sport sport;

public class Comfort {

@SerializedName("txt") public String info;

}

public class CarWash {

@SerializedName("txt") public String info;

}

public class Sport {

(aSerializedName( "txt") public String info;

}

}

daily\_fo「ecast中的具 体内容如下所示：

"dailyforecast":[

{-

date”：”2016-08-08”，

"cond":{

阵雨”

},

"trnp":{

"max":"34%

}

},

{

"cond":{

"txtd\*':"多云”

}, ~

“max”：“35”，

“min”：”29”

}

},

}…

daily\_fo「ecast中包含的是一个数组，数组中的每一项都代表着未来一天的天气信息。针对于这种情况，我们只需要定义出单日天气的实体类就可以了，然后在声明实体类引用的时候使用集合类型来进行声明。

在gson包下新建一个Forecast类，代码如下所示：

public class Forecast {

public String date;

(aSeriatizedName( "tmp")

public Temperature temperature;

@Se rializedName (" cond '\*)

public More more;

public class Temperature (

public String max;

public String min;

}

public class More {

@SerializedName("txtd")

public String info;

}

}

这样我们就把basic, aqi、now、suggestion和daily\_forecast对应的实体类全部都创建好了，接下来还需要再创建一个总的实例类来引用刚刚创建的各个实体类。在gson包下新建一个Weather类，代码如下所示：

public class Weather {

public String status;

public Basic basic;

public AQI aqi;

public Now now;

public Suggestion suggestion;

(QSe rializedName ("dailyfo recast") public List<Forecast> forecastList;

}

在 Weather 类中，我们对 Basic、AQI、Now、Suggestion 和 Forecast 类进行了引用。其中，由于daily forecast中包含的是一个数组，因此这里使用了 List集合来引用Forecast类。 另外，返回的天气数据中还会包含一项status数据，成功返回ok,失败则会返回具体的原因，那么这里也需要添加一个对应的status字段。

**2.3.3.2 编写天气界面**

创建一个用于显示天气信息的活动。右击。New->Activity—> Empty Activity,创建一个 WeatherActivity,并将布局名指定成 activity weather.xmlo。

由于所有的天气信息都将在同一个界面上显示，因此activity weather.xml会是一个很长的布 局文件。那么为了让里面的代码不至于混乱不堪，我们将界面的不同部分写在不同的布局文件里面，再通过引入布局的方式集成到activity, weather.xml中，这样整个布局文件就会显得非常工整。

右击res/layout—>New—^Layout resource file,新建一个title.xml作为头布局，代码如下所示：

<RelativeLayout

xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="match\_parent" android:layout\_height="?attr/actionBarSize">

<TextView

android: id="(a+id/title\_city"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:textColor=n#fff"

android:textSize="20sp" />

<TextView

android:id="@+id/title\_update\_time" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_marginRight="10dp" android: layout\_aHgnParentRight="true" android:layout\_centerVertical="true" android:textColor="#fff" android:textSize="16sp"/>

</RelativeLayout>

这段代码中，头布局中放置了两个TextView, 一个居中显示城市名，一个居右显示更新时间。

然后新建一个now.xml作为当前天气信息的布局，代码如下所示：

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/and roid" android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layoutmargin="15dp">

<TextView

android:id="@+id/degreetext"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="end"

android:textColor="#fff"

android:textSize="60sp" />

<TextView

android :id=,,(a+id/weather\_info\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layoutg ravity=u end"

android:textColor="#fff"

android:textSize="20sp" />

</LinearLayout>

当前天气信息的布局中也是放置了两个TextView, 一个用于显示当前气温，一个用于显示天气概况。

然后新建forecast.xml作为未来几天天气信息的布局，代码如下所示：

<LinearLayout

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

android:orientation="vertical"

android: layoiit\_width="match\_parent"

android:layout\_height="wrap\_content"

android: l.ayout\_margin=,l15dp"

android:background="#8000">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginLeft="i5dp"

android:layout\_marginTop="15dp"

android :text="^ 报”

android:textColor="#fff"

android:textSize="20sp"/>

<LinearLayout

android:id="@+id/forecast\_layout"

android:orientation="vertical"

android:layout\_width="match\_parent"

and roid: 'layout\_height="wrap\_content ">

</LinearLayout>

</LinearLayout>

这里最外层使用LinearLayout定义了一个半透明的背景，然后使用TextView定义了一个标 题，接着又使用一个LinearLayout定义了一个用于显示未来几天天气信息的布局。不过这个布局 中并没有放入任何内容，因为这是要根据服务器返回的数据在代码中动态添加的。

为此，我们还需要再定义一个未来天气信息的子项布局，创建forecast item.xml文件，代码 如下所示：

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

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="15dp">

<TextView

android: id="(a+id/date\_text"

android:tayout\_width="0dp"

android:layout\_height="wrap\_content"

android:layoutgravity="cente r\_ve rtical"

android:layout\_weight="2"

android:textColor="#fff"/>

<TextView

android: id="(a+id/info\_text"

android :tayout\_width=,,0dp"

android:layout\_height="wrap\_content"

android:layoutgravity="center\_vertical"

android:layout\_weight="1"

android:gravity="center" android:textColor="#fff"/>

<TextView

android:id="@+id/max\_text"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layoutgravity="center"

and roid :layout\_weight=,,l,,

android:gravity="right" android: textColor=,,#fff,,/>

<TextView

android: id="(a+id/min\_text"

android: layout\_width=\*,0dp"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_weight="l"

android:gravity=H right"

android:textColor="#fff"/>

</LinearLayout>

子项布局中放置了 4个TextView, 一个用于显示天气预报日期，一个用于显示天气概况，另 外两个分别用于显示当天的最高温度和最低温度。

然后新建aqi.xml作为空气质量信息的布局，代码如下所示：

<LinearLayout

xmlns:and roid="http://schemas.android.com/apk/res/android" android:orientation=,•vertical"

android: layout\_width=,,match\_parent" android: layout\_height="wrap\_content,1 android:layout\_margin="15dp"

android:background="#8000">

<TextView

android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout marginLeft="15dp" android:layout\_marginTop="15dp" android :text="^ 气质量” android:textColor="#fff" android: textSize=,,20sp"/>

<LinearLayout

android: ■Layout\_width="match\_parent" android:layout\_height="wrap\_content" android:layout\_margin="15dp">

<RelativeLayout

android :layout\_width=,,0dp"

android:layout\_height="match\_parent"

android:layout\_weight="1">

<LinearLayout

android:orientation="vertical" android: layout\_width=,,match\_parent" android:layout\_height="wrap\_content" android:layoutcenterlnPa rent="t rue">

<TextView

android:id="@+id/aqi\_text" android :layout\_width=,,wrap\_content" android:layout\_height="wrap\_content" android:layout\_gravity="center" android:textCoTor="#fff"

android:textSize="40sp"

/>

<TextView

android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_gravity="center" android :text="AQI 指数” android:textColor="#fff"/>

</LinearLayout>

</RelativeLayout>

<RelativeLayout

android:layout\_width="Odp" android:layout\_height="match\_parent"

android:layout\_weight="l">

<LinearLayout

android:orientation="vertical"

android:layout\_width="match\_parentH

android :layout\_height=,,wrap\_content"

and roid: layoutcenterlnPa rent=,,true">

<TextView

and roid: id="(a+id/pm25\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layoutgravity="center"

android:textCoTor="#fff"

android:textSize="40sp"

/>

<TextView

android:layout\_width="wrap\_content"

and roid :layout\_height="wrap\_content,,

android:layout\_gravity="center"

android:text="PM2.5 指数”

android:textColor="#fff"

/>

</LinearLayout>

</RelativeLayout>

</LinearLayout>

</LinearLayout>

这个布局中的代码中使用 LinearLayout定义了一个半透明的背景，然后使用TextView定义了一个标题。接下来，使用 LinearLayout和RelativeLayout嵌套的方式实现了一个左右平分并且居中对齐的布局，分别用于显示AQI指数和PM2.5指数。

然后新建suggestion.xml作为生活建议信息的布局，代码如下所示:

<LinearLayout

xmlns:and roid="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_pa rent"

android:layout\_height="wrap\_content"

android:layout\_margin="15dp"

android:background="#8000">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android :layout\_marginLeft=,'15dp"

android :layout\_marginTop=,,15dp"

and roid: text="± 活建议"

android:textColor="#fff"

android:textSize="20sp"/>

<TextView

android: id=,,(a+id/comfort\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="15dp"

android:textCoTor="#fff" />

<TextView

android: id="(a+icl/car\_wash\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="15dp"

android:textCoTor="#fff" />

<TextView

android :id=,,@+id/sport\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="15dp"

and roid:textCoTor="#fff" />

</LinearLayout>

这里同样也是先定义了一个半透明的背景和一个标题，然后下面使用了 3个TextView分别 用于显示舒适度、洗车指数和运动建议的相关数据。

这样我们就把天气界面上每个部分的布局文件都编写好了，接下来的就是将它们引入到 activity weather.xml 当中，如下所示：

<FrameLayout

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

android:layout\_width="match\_parent"

android:layout\_height="match\_pa rent"

android: background=,,@color/colorPriniary,,>

<ScrollView

and roid: id=”@+id/weather\_layoiit"

and roid: layout\_width=,,match\_pa rent"

android:layout\_height="match\_parent"

android:scrollbars="none"

android:overScrollMode="never">

<LinearLayout

android:orientation="vertical"

android:layout\_width="match\_pa rent"

android:layout\_height="wrap\_content"> <include layout="@layout/title" />

<include layout="@layout/now" />

<include layout="@layout/forecast" />

<include layout="@layout/aqi" />

<include layout="@layout/suggestion" />

</LinearLayout>

</ScrollView>

</FrameLayout>

首先最外层布局使用了一个FrameLayout,并将它的背景色设置成colorPrimary0 然后在FrameLayout中嵌套了一个ScrollView,这是因为天气界面中的内容比较多，使用 ScrollView可以允许我们通过滚动的方式查看屏幕以外的内容。

由于ScrollView的内部只允许存在一个直接子布局，因此这里又嵌套了一个垂直方向的 LinearLayout,然后在LinearLayout中将刚才定义的所有布局逐个引入。

**2.3.3.3 将天气显示到界面上**

首先需要在Utility类中添加一个用于解析天气JSON数据的方法，如下所示：

public class Utility {

/\*\*

\*将返回的JSON数据解析成Weather实体类

\*/

public static Weather handleWeatherResponse(String response) { try {

JSONObject jsonObject = new JSONObject(response);

JSONArray jsonArray = j sonObject.getJSONArray("HeWeather"); String weatherContent = jsonArray.getJSONObject(O).toStringO; return new Gson().fromJson(weatherContent, Weather.class);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

可以看到，handleWeatherResponse()方法中先是通过JSONObject和JSONArray将天气 数据中的主体内容解析出来，即如下内容：

{"status": "ok",

"basic": {},

"aqi": {},

"now": {},

"suggestion": {},

"daily forecast":[]

}

然后由于之前已经按照上面的数据格式定义过相应的GSON实体类，因此只需要通过 调用fromJsonO方法就能直接将JSON数据转换成Weather对象了。

接下来的工作是我们如何在活动中去请求天气数据，以及将数据展示到界面上。修改WeatherActivity中的代码，如下所示：

public class WeatherActivity extends AppCompatActivity {

private Scrollview weatherLayout;

private Textview titleCity;

private Textview titleUpdateTime;

private Textview degreeText;

private Textview weatherlnfoText;

private LinearLayout forecastLayout;

private Textview aqiText;

private Textview pm25Text;

Override protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState);

setContentView(R.layout.activityweather);

//初始化各控件 —

weatherLayout = (Scrollview) findViewByld(R.id.weatherlayout); titleCity = (TextView) findViewById(R.id.titlecity); titleUpdateTime = (TextView) findViewByld(R.id.titleupdatetime); degreeText = (TextView) findViewByld(R.id.degree text); weatherlnfoText = (TextView) findViewByld(R.id.weather info text); forecastLayout = (LinearLayout) findViewById(R.id.forecastlayout); aqiText = (TextView) findViewByld(R.id.aqitext); pm25Text = (TextView) findViewByld(R.id.pm25\_text); comfortText = (TextView) findViewByld(R.id.comfort text); carWashText = (TextView) findViewByld(R.id.car wash text); sportText = (TextView) findViewById(R.id.sporttext);

SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences (this);

String weatherstring = prefs.getString("weather", null); if (weatherstring != null) {

//有缓存时直接解析天气数据

Weather weather = Utility.handleWeatherResponse(weatherString); showWeatherlnfo(weather);

} else {

//无缓存时去服务器查询天气

String weatherld = getlntent().getStringExtra("weatherid"); weatherLayout.setvisibility(View.INVISIBLE); —

requestweather(weatherld);

)

}

/\*\*

\*根据天气id请求城市天气信息

\*/

public void requestweather(final String weatherld) {

String weatherllrl = "http://guolin.tech/api/weather?cityid=" + weatherld + "&key=bc0418b57b2d4918819d3974acl285d9” ；

HttpUtil.sendOkHttpRequest(weatherllrl, new Callback。{ @0verride public void onResponse(Call call, Response response) throws lOException { final String responseText = response.body().st ring();

final Weather weather = Utility.handleWeatherResponse(responseText); runOnlliThread (new Runnable() {

(QOverride

public void run() {

if (weather != null && "ok".equals(weather.status)) { SharedPreferences.Editor editor = PreferenceManager. getDefaultSharedPreferences(WeatherActivity.this). edit();

editor.putString("weather", responseText);

editor.apply(); showWeatherlnfo(weather);

} else {

Toast.makeText (WeatherActivity. this,"获取天气信息失败”, Toast.LENGTH\_SHORT).show();

}

}

})；

}

public class ChooseAreaFragment extends Fragment {

(Override在onCreateO方法中仍然先是去获取一些控件的实例，然后会尝试从本地缓存中读取天气数据。那么第一次是没有缓存的，因此就会从Intent中取出天气id,并调用requestWeather()方法来从服务器请求天气数据。在请求数据的时候先将ScrollView进行隐藏，不然空数据的界面看上去会很奇怪。

requestWeatherO方法中先是使用了参数中传入的天气id和我们之前申请好的API Key拼装出一个接口地址，接着调用Httplltil. sendOkHttpRequest ()方法来向该地址发出请求，服务器会将相应城市的天气信息以JSON格式返回。然后我们在onResponseO回调中先调用Utility. handleWeatherResponseO方法将返回的JSON数据转换成Weather对象，再将当前线程切换到主线程。然后进行判断，如果服务器返回的status状态是ok,就说明请求天气成功了，此时将返回的数据缓存到SharedPreferences当中，并调用showWeatherlnfo()方法来进行内容显示。showWeatherlnfoO方法中的逻辑就比较简单了，其实就是从Weather对象中获取数据，然后显示到相应的控件上。注意在未来几天天气预报的部分我们使用了一个for循环来处理每天的天气信息，在循环中动态加载forecast\_item.xml布局并设置相应的数据，然后添加到父布局当中。设置完了所有数据之后，记得要将ScrollView重新变成可见。

这样我们就将首次进入WeatherActivity时的逻辑全部梳理完了，那么当下一次再进入 WeatherActivity时，由于缓存已经存在了，因此会直接解析并显示天气数据，而不会再次发起网络请求了。

处理完了 WeatherActivity中的逻辑，接下来我们要做的，就是如何从省市县列表界面跳转到 天气界面了，修改ChooseAreaFragment中的代码，如下所示：

public void onActivityCreated(Bundle savedlnstanceState) { super.onActivityCreated(savedlnstanceState); listview.setOnItemClickListener(new Adapterview.OnItemClickListener() { (QOverride

public void onItemClick(AdapterView<?> parent, View view, int position, long id) { if (currentLevel == LEVELPROVINCE) {

selectedProvince = provinceList.get(position); queryCities();

} else if (currentLevel == LEVELCITY) { selectedCity = cityList.get(position); queryCounties();

) else if (currentLevel == LEVEL\_COUNTY) {

String weatherld =. countyList.get(position).getWeatherld(); Intent intent = new Intent(getActivity(), WeatherActivity. class);

intent.putExtra("weather^id", weatherld); startActivity(intent); getActivity().finish();

}

}

})；

}…

这里在onItemClick()方法中加入了一个if判断，如果当前级别是LEVEL\_ COUNTY,就启动WeatherActivity,并把当前选中县的天气id传递过去。

另外，我们还需要在MainActivity中加入一个缓存数据的判断才行。修改MainActivity中的 代码，如下所示：

public class MainActivity extends AppCompatActivity {

@0verride

protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState); setContentView(R.layout.activitymain);

SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences (this);

if (prefs.getString("weather", null) != null) {

Intent intent = new Intent(this, WeatherActivity.class); sta rtActivity(intent); finishO;

}

}

}

这里在onCreate()方法的一开始先从SharedPreferences文件中读取缓存数据， 如果不为null就说明之前已经请求过天气数据了，那么就没必要让用户再次选择城市，而是直 接跳转到WeatherActivity即可。

**2.3.3.4 获取必应每日一图**

每日一图的图片都是由必应精挑细选出来的，并且每天都会变化，如果我们使用它们来作为天气界面的背景图，不仅可以让界面变得更加美观，而且解决了界面一成不变、过于单调的问题。

为此访问别人架设的专门的一个获取必应每日一图的接口 ： http://guolin.tech/api/bing\_pico

访问这个接口，服务器会返回今日的必应背景图链接：

http://cn.bing.com/az/hprichbg/rb/ ChicagoHarborLH\_ZH-CN9974330969\_1920xl080.jpg。

修改activity\_weather.xm 1中的代码, 如下所示：

<FrameLayout

xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="match\_parent"

android:layout\_height="match\_parent" android: background="@color/colorPrimary,,>

<ImageView

android:id="@+id/bing\_pic\_img" android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:scaleType="centerCrop" />

<ScrollView

android:id="@+id/weathe「layout"

android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:scrollbars="none"

android:overScrollMode="never">

</ScrollView>

</FrameLayout>

这里我们在FrameLayout中添加了一个Image View,并且将它的宽和高都设置成match\_ parento由于FrameLayout默认情况下会将控件都放置在左上角，因此ScrollView会完全覆盖住 Image View,从而Image View也就成为背景图片了。

接着修改WeatherActivity中的代码，如下所示：

public class WeatherActivity extends AppCompatActivity {

private Imageview bingPidmg;

^Override

protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState); setContentView(R.layout.activityweather);

//初始化各控件 —

bingPidmg = (ImageView) findViewById(R.id.bing\_pic\_img);

String bingPic = prefs.getString("bing\_\_pic", null); if (bingPic != null) {

Glide.with(this).load(bingPic).into(bingPidmg); } else {

loadBingPic();

}

}

/\*\*

\*根据天气id请求城市天气信息

\*/

public void requestweather(final String weatherld) {

loadBingPicO;

/\*\*

\*加载必应每日一图

\*/

private void LoadBingPic() {

String requestBingPic = "http://guolin.tech/api/bing\_pic"; HttpUtil.sendOkHttpRequest(requestBingPicF new Callbackf) { ^Override public void onResponse(Call call., Response response) throws lOException { final String bingPic = response.body().stringO; SharedPreferences.Editor editor = PreferenceManager. getDefaultSharedPreferences(WeatherActivity.this).edit(); editor.putString("bing\_pic", bingPic);

editor.apply(); runOnUiThread(new RunnableO { (dOverride public void run() {

Glide.with(WeatherActivity.this).load(bingPic).into (bingPidmg);

^Override public void e.printStackTrace();

} })； }

首先在onCreate()方法中获取了新增控件ImageView的实例，然后尝试从 SharedPreferences中读取缓存的背景图片。如果有缓存的话就直接使用Glide来加载这张图片，如果没有的话就调用loadBingPic()方法去请求今日的必应背景图。

loadBingPic()方法中的逻辑：先是调用了 HttpUtil.sendOkHttpRequest() 方法获取到必应背景图的链接，然后将这个链接缓存到SharedPreferences当中，再将当前线程切换到主线程，最后使用Glide来加载这张图片就可以了。另外需要注意，在requestWeatherO 方法的最后也需要调用一下loadBingPic()方法，这样在每次请求天气信息的时候同时也会刷新背景图片。

修改WeatherActivity 中的代码，将背景图和状态栏融合到一起，如下所示：

public class WeatherActivity extends AppCompatActivity {

@0verride protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState);

if (Build.VERSION.SDK\_INT >= 21) {

View decorView = getWindow().getDecorView(); decorView.setSystemUiVisibilityf View.SYSTEM\_UI\_FLAG\_LAYOUT\_\_FULLSCREEN I View. SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE);

getWindowO.setStatusBarColor(Color.TRANSPARENT); }

setContentView(R.layout.activityweather);

}

}

由于这个功能是Android 5.0及以上的系统才支持的，因此我们先在代码中做了一个系统版本号的判断，只有当版本号大于或等于21,也就是5.0及以上系统时才会执行后面的代码。

接着我们调用了 getWindow() ,getDecorView()方法拿到当前活动的DecorView,再调用它 的setSystemUiVisibility()方法来改变系统 UI的显示，这里传入View.SYSTEM\_UI\_ FLAG\_LAYOUT\_FULLS和View. SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE 就表示活动的布局会显示在状态栏上面，最后调用一下setStatusBarColor()方法将状态栏设置成透明色。

这些代码就可以实现让背景图和状态栏融合到一起的效果了。不过，发现还是有些问题，天气界面的头布局几乎和系统状态栏紧贴到一起了。

这是由于系统状态栏已经成为我们布局的一部分，因此没有单独为它留出空间。我们借助android:fitsSystemWindov/s属性就可以了。修改activity一 weather.xml中的代码，如下所示：

<FrameLayout

xmlns:android="http://schemas.android.com/apk/res/android" android: layout\_width=,,match\_parent"

android:layout\_height="match\_pa rent"

android:background="@color/colorPrimary">

vScrollView

android:id="@+id/weather\_layout" android:layout\_width="match\_parent" android:layout\_height="match\_parent" android: scroHbars="none"

android:overScrollMode="never">

<LinearLayout

android:orientation="vertical" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android: fitsSystemWindows="true,,>

</LinearLayout>

</ScrollView>

</FrameLayout>

这里在 ScrollView 的 LinearLayout 中增加了 android: fitsSystemWindows 属性，设置成 true就表示会为系统状态栏留出空间。

**2.3.4 手动更新天气和切换城市**

经过以上阶段的开发，现在天气APP的主体功能已经有了，不过我们发现目前存在着一个比较严重的bug,就是当你选中了某一个城市之后，就没法再去查看其他城市的天气了，即使退出程序，下次进来的时候还会直接跳转到Weather Activity。

因此，在第下一步中我们要加入切换城市的功能，并且为了能够实时获取到最新的天气，我们还会加入手动更新天气的功能。

**2.3.4.1 手动更新天气**

先来实现一下手动更新天气的功能。由于我们在上一节中对天气信息进行了缓存，目前每次展示的都是缓存中的数据，因此现在非常需要一种方式能够让用户手动更新天气信息。

这里我准备采用下拉刷新的方式触发更新事件。首先修改activity\_weather.xml中的代码，如下所示：

<FrameLayout

xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="matchparent" android: layout\_height=,,match\_parent" android: background=,,(acolor/colorPrimary">

ondroid・ support.v4.widget.SwipeRefreshLayout android:id="@+id/swipe\_refresh" and roid: l.ayout\_\_width=,,match\_pa rent" android:layout\_height="match\_parent">

<ScrollView

android: id="(a+id/weather\_'layout" android:layout\_width="matchparent" android:layout\_height="match\_parent" android: scroflbars="none"

android:overScrollMode="never">

</ScrollView>

</android. support.v4・ widget.SwipeRefreshLayout>

</FrameLayout>

可以看到，这里在ScrollView的外面又嵌套了一层SwipeRefreshLayout,这样ScrollView就 自动拥有下拉刷新功能了。

然后修改WeatherActivity中的代码，加入更新天气的处理逻辑，如下所示：

public class WeatherActivity extends AppCompatActivity {

public SwipeRefreshLayout swipeRefresh;

(QOverride

protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState);

swipeRefresh = (SwipeRefreshLayout) findViewByld(R.id.swipe\_\_refresh); swipeRefresh.setColorSchemeResources(R.color.colorPrimary); SharedPreferences prefs = PreferenceManager.

getDefaultSharedPreferences(this);

String weatherString = prefs.getString("weather", null); final String weatherld;

if (weatherString != null) {

//有缓存时直接解析天气数据

Weather weather = Utility.handleWeatherResponse(weatherString); weatherld = weather.basic.weatherld;

showWeatherlnfo(weather);

} else {

//无缓存时去服务器查询天气

weatherld = getlntent().getSt ringExt ra("weather\_id"); weatherLayout.setVisibility(View.INVISIBLE); requestweather(weatherld);

}

swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.

OnRefreshListener() {

QOverride

public void onRefresh() { requestweather(weatherld);

} })；

}

/\*\*

\*根据天气id请求城市天气信息

\*/

public void requestWeather(final String weatherld) {

String weatherUrl = "http://guolin.tech/api/weather?cityid=" + weatherld + ,,&key=bc0418b57b2d4918819d3974acl285d9,';

HttpUtil. sendOkHttpRequest (weatherllrl, new Callback!) { (QOverride

public void onResponse(Call call, Response response) throws lOException {

runOnUiThread(new Runnable() {

^Override

public void run() (

if (weather != null && "ok".equals(weather.status)) { SharedPreferences.Editor editor = PreferenceManager. getDefaultSharedPreferences(WeatherActivity.

this).edit();

editor.putString("weather", responseText); editor.apply();

showWeatherlnfo(weather);

} else {

Toast.makeText (WeatherActivity. this,"获取天气信息失败", Toast,LENGTH\_SHORT).show();

}

swipeRefresh.setRefreshing(false);

}

})；

}

^Override

public void onFailure(Call call, lOException e) {

e.printStackTrace();

runOnUiThread(new Runnable() {

^Override

public void run() {

Toast.makeText(WeatherActivity.this,"获取天气信息失败”， Toast.LENGTHSHORT).show();

swipeRefresh.setRefreshing(false);

)

})；

}

})；

loadBingPic();

首先在onCreate()方法中获取到了 SwipeRefreshLayout的实例，然后调用setColorSchemeResources()方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的colorPrimary作为进度条的颜色了。接着定义了一个weatherld变量，用于记录城市的天气 id,然后调用setOnRef reshListener()方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候，就会回调这个监听器的onRefresh()方法，我们在这里去调用requestWeather()方法请求天气信息就可以了。

另外，当请求结束后，还需要调用SwipeRefreshLayout的setRef reshing ()方法并传入false,用于表示刷新事件结束，并隐藏刷新进度条。

**2.3.4.2 切换城市**

要切换城市，那么就需要遍历全国省市县的数据，而这个功能我们早在前面就已经完成了，并且当时考虑为了方便后面的复用，特意选择了在碎片当中实现。因此，我们其实只需要在天气界面的布局中引入这个碎片，就可以快速集成切换城市功能了。

虽说实现原理简单，但是我们也不可能让引入的碎片把天气界面遮挡住，将碎片放入到滑动菜单中，正常情况下它不占据主界面的任何空间，想要切换城市的时候只需要通过滑动的方式将菜单显示出来就可以了。

首先按照Material Design，我们需要在头布局中加入一个切换城市的按钮，不然的话用户可能根本就不知道屏幕的左侧边缘是可以拖动的。修改 title.xml中的代码，如下所示：

<RelativeLayout

xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="match\_parent"

android:layout\_height="?att r/actionBa rSize">

<Button

android: id=,,@+id/nav\_buttonu

android: layout\_\_width=”30dp"

android:layout\_height="30dp" android: Layout\_marginLef t=',10dp" android:layout\_alignPa「entLeft=”true" android :Layout\_\_centerVertical="t rue" android: background=,,@drawable/ic\_home" />

</RelativeLayout>

这里添加了一个Button作为切换城市的按钮，并且让它居左显示。提前准备好了一张图片来作为按钮的背景图。

接着修改activity weather.xml布局来加入滑动菜单功能,如下所示：

<FrameLayout

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

android: ■Layout\_width="match\_parent" android: layout\_height=,,match\_parent" android: background="(acolor/colorPrimary">

<android・ support.v4.widget.DrawerLayout android: i d="@+id/d rawe r\_\_layout" android: layout\_width=,'match\_\_parent" android:layout\_height="match\_parent">

<android.support.v4.widget.SwipeRefreshLayout android:id="@+id/swipe\_refresh" android:layout\_width="match\_parent" android:layout\_height="match\_parent">

</android.support.v4.widget.SwipeRef reshLayout>

<fragment

android:id="@+id/choose\_area\_fragment"

android: name=licom ・ coolweather. android ・ ChooseAreaFragment11 android:Layout\_width="match\_parent" android :layout\_height=,,match\_parent" android:layout\_g ravity="start"

/> ~

</android, support.v4.widget.DrawerLayout>

</FrameLayout>

可以看到，我们在SwipeRefreshLayout 的外面又嵌套了一层 DrawerLayouto DrawerLayout 中的第一个子控件用于作为主屏幕中显示的内容，第二个子控件用于作为滑动菜单中显示的内 容，因此这里我们在第二个子控件的位置添加了用于遍历省市县数据的碎片。

接下来需要在WeatherActivity中加入滑动菜单的逻辑处理，修改WeatherActivity中的代码, 如下所示：

public class WeatherActivity extends AppCompatActivity {

public DrawerLayout drawerLayout;

private Button navButton;

^Override

protected void onCreate(Bundle savedlnstanceState) { super.onCreate(savedlnstanceState);

drawerLayout = (DrawerLayout) findViewById(R.id.drawer\_\_layout); navButton = (Button) findViewByld(R.id.nav\_\_button);

navButton.setOnClickListener(new View.OnClickListener() { @0verride public void onCLick(View v) {

drawerLayout.openDrawer(GravityCompat.START);

}

})；

}

}

首先在onCreate()方法中获取到新增的DrawerLayout和Button的实例，然后在 Button的点击事件中调用DrawerLayout的openDrawer()方法来打开滑动菜单就可以了。

这仅仅是打开了滑动菜单，我们还需要处理切换城市后的逻辑。这就必须要在ChooseAreaFragment中进行了，因为之前选中了某个城市后是跳转到WeatherActivity的,而现在由于我们本来就是在WeatherActivity当中的，因此并不需要跳转, 只是去请求新选择城市的天气信息就可以了。

那么这里需要根据ChooseAreaFragment的不同状态来进行不同的逻辑处理，修改ChooseAreaFragment中的代码，如下所示：

public class ChooseAreaFragment extends Fragment {

(QOverride public void onActivityCreated(Bundle savedlnstanceState) { super.onActivityCreated(savedlnstanceState);

listView.setOnItemClickListener(new Adapterview.OnItemClickListener() { ^Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) ( if (currentLevel == LEVEL\_PROVINCE) {

selectedProvince = provinceList.get(position); queryCities();

} else if (currentLevel == LEVELCITY) { selectedCity = cityList.get(position); queryCounties();

} else if (currentLevel == LEVELCOUNTY) {

String weatherld = countyList.get(position).getWeatherId();

if (getActivity() instanceof MainActivity) {

Intent intent = new Intent(getActivity(), WeatherActivity. class);

intent.putExtra("weatherid", weatherld); startActivity(intent);

getActivity(),finish();

} else if (getActivityO instanceof WeatherActivity) { WeatherActivity activity = (WeatherActivity) getActivityO; activity.drawerLayout ・ closeDrawers(); activity.swipeRefresh・ setRefreshing(true); activity.requestweather(weatherld);

}

}

}

})；

}

这样我们就把切换城市的功能全部完成了。

**2.3.5 后台自动更新天气**

为了要让天气APP更加智能，在第五阶段我们准备加入后台自动更新天气的功能，这样就可以尽可能地保证用户每次打开软件时看到的都是最新的天气信息。

要想实现上述功能，就需要创建一个长期在后台运行的定时任务。首先在 service 包下新建一个服务，右击 com.coolweather.android.service—>New—^Service—> Service,创建一个AutoUpdateService,并将Exported和Enabled这两个属性都勾中。然后修改 AutoUpdateService中的代码，如下所示：

public class AutoUpdateService extends Service {

^Override

public IBinder onBind(Intent intent) { return null;

}

^Override

public int onStartCommand(Intent intent, int flags, int startld) ( updateWeather(); updateBingPic();

AlarmManager manager = (AlarmManager) getSystemService(ALARMSERVICE); int anHour = 8 \* 60 \* 60 \* 1000; // 这是 8 小时的毫秒数 —

long triggerAtTime = SystemClock.elapsedRealtime() + anHour; Intent i = new Intent(this, AutoUpdateService.class);

Pendingintent pi = Pendingintent.getService(this, 0# i, 0); manager.cancel(pi);

manager.set(AlarmManager.ELAPSEDREALTIMEWAKEUP, triggerAtTime, pi); return super.onStartCommand(intent, flags, startld);

}

\*更新天气信息

\*/

private void updateWeather(){

SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this); String weatherstring = prefs.getString("weather", null);

if (weatherstring != null) (

//有缓存时直接解析天气数据

Weather weather = Utility.handleWeatherResponse(weatherString);

String weatherld = weather.basic.weatherld;

String weatherllrl = "http://guolin.tech/api/weather?cityid=" +

weatherld + ,,&key=bc0418b57b2d4918819d3974acl285d9n;

HttpUtil.sendOkHttpRequest(weatherllrl, new Callback。{ ^Override

public void onResponse(Call call, Response response) throws lOException {

String responseText = response.body().stringO;

Weather weather = Utility.handleWeatherResponse(responseText); if (weather != null && "ok".equals(weather.status)) ( SharedPreferences.Editor editor = PreferenceManager. getDefaultSharedPreferences (AutollpdateService.this). edit();

editor.putString("weather", responseText); editor.apply();

}

} ^Override

public void onFailure(Call call, lOException e) ( e.printStackT race();

}

})；

/\*\*

\*更新必应每日一图

\*/

private void updateBingPic() {

String requestBingPic = "http://guolin.tech/api/bing\_pic"; HttpUtil.sendOkHttpRequest(requestBingPic, new Callback。 { (QOverride

public void onResponse(Call call, Response response) throws lOException { String bingPic = response.body().string();

SharedPreferences.Editor editor = PreferenceManager.getDefault

Sha redPreferences(AutoUpdateService.this).edit(); editor.putString("bing pic", bingPic); editor.apply();

} ^Override

public void onFailurefCall call, lOException e) {

e.printStackTrace();

}

})；

}

}

在onStartCommandO方法中先是调用了 updateWeather()方法来更新天气， 然后调用了 updateBingPicO方法来更新背景图片。这里我们将更新后的数据直接存储到 SharedPreferences文件中就可以了，因为打开WeatherActivity的时候都会优先从SharedPreferences 缓存中读取数据。

之后就是创建定时任务了，将时间间隔设置为8小时，8小时后AutoUpdateReceiver的onStartCommand ()方法就会重新执行， 这样也就实现后台定时更新的功能了。

不过，我们还需要在代码某处去激活AutoUpdateService这个服务才行。修改WeatherActivity 中的代码，如下所示：

public class WeatherActivity extends AppCompatActivity {

/\*\*

\*处理并展示Weather实体类中的数据。

\*/

private void showWeatherlnfo(Weather weather) (

if (weather != null && "ok".equals(weather.status)) (

Intent intent = new Intent(this, AutoUpdateService.class); startservice(intent);

} else (

Toast.makeText (WeatherActivity. this,"获取天气信息失败”，Toast. LENGTH\_ SHORT).show(); ~

}

}

}

这里在showWeather()方法的最后加入启动AutoUpdateService这个服务的代码， 这样只要一旦选中了某个城市并成功更新天气之后，AutoUpdateService就会一直在后台运行，并保证每8小时更新一次天气。

**2.3.6 修改图标和名称**

将图片命名成logo.png,放入到所有以mipmap开头的目录下，然后修改AndroidManifest.xml中的代码，如下所示：

<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.coolweather.android">

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

<application

android:name="org.litepal.LitePalApplication"

android:allowBackup="true"

android: icon=,l@mipmap/'logo,>

and roid: label="(ast ring/appname" android:supportsRtl="true" and roid: theme="@sty'le/AppTheme">

</application>

</manifest>

这里将<application>标签的android:icon属性指定成@mipmap/logo就可以修改程序图标了。接下来我们还需要修改一下程序的名称，打开res/values/string.xml文件，其中app name 对应的就是程序名称，将它修改即可，如下所示：

<resources>

<string name="app\_name”>天气v/string〉

</resources>

**2.3.7 用sqllite实现注册登录**

4测试方案

（小四号 宋体 1.5倍行距）

5 用户手册

（小四号 宋体 1.5倍行距）

6 项目总结

小组沟通不完善，时间过紧，部分人员未进行过Android编程开发，编程人员代码能力不足，导致功能实现并不完全，出现了虎头蛇尾的现象。在移动互联时代，智能手机已成为我们日常最重要的随身物品，给我们的工作和生活带来了极大的便利,天气预报手机App作为一款提供天气资讯的最常用软件,功能强大、内容丰富,可以帮助人们及时查询天气信息，满足使用者者合理安排出行的需求。Android平台的兼容性和开放性极强，本文实现了在安卓平台上开发一款查看实时和未来天气的天气预报软件。基于Android的天气预报APP在Android真机网络畅通的情况下进行了测试，达到了预期的运行效果。能够轻松便捷地通过APP获取全国各地城市的天气信息。除了具有显示天气的功能外，我们需要在日后的工作中不断探索和实践,以建立更实用的天气预报软件。当然,本软件还有很多功能可以改进和创新,大多数天气预报app只注重显示天气的功能,没有对用户的数据进一步分析利用,未来可以对用户每日天气情况、位置进行收集,增加更加人性的功能。如:一周的天气情况，通过折线图等方式形象的展示出来;评选出最佳天气周,在天气好得时候推送旅游线路;用户界面也可以随着天气情况来改变,污染严重的时候是黑色，天气好得时候是蓝色等。使用户交互更加友好，这也是我日后需要不断完善的方向。

最后，我们小组成员在进行了为期2周的移动应用开发，从零开始，一点点的进行分析，为了有些功能进行反复的探讨，并不断改进。

本次实验使用了大部分的专业课内容，包括数据库的设计与实现，安卓代码编程，还是信息系统软件的分析与设计在内的，都或多或少的在此有所体现，记得当时谢益武谢老师有讲过一句话，学是为了用，本次实验确确实实得到了印证。

感谢小组中每一个成员这两周以来的付出和努力，也感谢给小组提供帮助的同学，朋友。最后，希望能够做好系统进一步的完善，实现更好的应用！谢谢！