**南 京 师 范 大 学**

**《移动平台开发》**

**实验报告**



**题 目： 基于Android的天气预报软件**

**学 院： 计算机科学与技术学院**

**专 业： 软件工程**

**姓 名： 钱迪琦**

**学 号： 19160229**

**任课教师： 黄为民**

**计算机科学与技术学院 制**

**二○一八 年 十二 月 十六 日**

**目录**

[1 实验要求 1](#_Toc532763477)

[2 实验环境 1](#_Toc532763478)

[3 实现技术 1](#_Toc532763479)

[3.1 服务器信息分析 1](#_Toc532763480)

[3.1.1 省、市、县数据 1](#_Toc532763481)

[3.1.2 天气数据 2](#_Toc532763482)

[3.1.3 获取数据 4](#_Toc532763483)

[3.2 罗列全国所有省、市、县信息 4](#_Toc532763484)

[3.2.1 数据库存储省、市、县 4](#_Toc532763485)

[3.2.2 显示省、市、县 6](#_Toc532763486)

[3.3 显示天气信息 10](#_Toc532763487)

[3.3.1 解析天气数据 10](#_Toc532763488)

[3.3.2 显示天气数据 11](#_Toc532763489)

[3.4 下拉刷新 13](#_Toc532763490)

[3.5 切换城市 14](#_Toc532763491)

[3.6 自动更新 15](#_Toc532763492)

[4 运行示例 16](#_Toc532763493)

[5 收获与体会 18](#_Toc532763494)

# 1 实验要求

利用Android平台开发一个天气预报软件，要求能够查看全国任意城市的天气信息，能够自动切换城市，提供手动更新和后台自动更新等功能。

# 2 实验环境

操作系统：Windows 10

开发平台：Android Studio 3.2

JRE: 1.8.0\_152-release-1136-b06 amd64

JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

# 3 实现技术

## 3.1 服务器信息分析

本程序采用了和风天气提供的API，但是和风天气服务器返回的内容过于详细，不利于我们解析，而CSDN的一位博主对和风天气API进行了二次包装，简化了访问方法，已经满足了本程序的要求。所以本程序直接采用此博主的服务器接口获取数据，并将数据进行了筛选。

获取的数据分为两个部分，一个是全国所有的省、市、县数据，还有一个部分是具体城市的天气情况，服务器返回的所有数据均为JSON格式。

### 3.1.1 省、市、县数据

访问如下地址<http://guolin.tech/api/china>即可获取全国所有省份的数据，数据格式如下：

[{"id":1,"name":"北京"},{"id":2,"name":"上海"}...]

获取某一省份下所有市的数据只需要取出该省份的id，将id添加到url地址的最后面即可，例如江苏省的id为16，则只需访问<http://guolin.tech/api/china/16>即可获取江苏省所有市的数据。得到的数据格式如下：

[{"id":113,"name":"南京"},{"id":114,"name":"无锡"}...]

不难发现市级的数据格式与省级相似，同样取出想要查看的市的id，将其添加到url地址的最后就可以查看该市下所有县的数据，例如南京市的id是113，则只需要访问<http://guolin.tech/api/china/16/113>即可获取南京市所有县级行政单位的数据。得到的数据格式如下：

[{"id":921,"name":"南京","weather\_id":"CN101190101"}…]

通过这里的weather\_id我们就能获取该地区的具体天气信息。

### 3.1.2 天气数据

通过注册一个和风天气账户，我们可以通过访问如下地址来获取具体城市的天气数据：http://guolin.tech/api/weather?cityid=CN101190101&key=...这里的cityid填写城市的weather\_id，而key填写登陆注册的和风天气账户获取的API key，获取的天气数据格式如下：

{

"HeWeather":[

{

"status":"ok",

"basic":{},

"aqi":{},

"now":{},

"suggestion":{},

"daily\_forecast":[]

}

]

}

其中status表示请求的状态，ok表示成功。basic中包含城市的一些基本信息，aqi中包含了当前的空气质量，now中包含了当前的天气信息，suggestion中包含了一些相关的生活建议，daily\_forecast中包含了未来几天的天气情况。

basic的格式如下，update中的loc表示更新的时间。

"basic":{

"city":"南京",

"id":"CN101190401",

"update":{

"loc":" 2018-12-16 18:46"

}

}

aqi的格式如下。

"aqi":{

"city":{

"aqi":" 118",

"pm25":"89"

}

}

now的格式如下。

"now":{

"tmp":"6",

"cond":{

"txt":"多云"

}

}

suggestion的格式如下。

"suggestion":{

"comf":{

"txt":"今天夜间天气晴好，但仍会使您感觉偏冷，不很舒适，请注意适时添加衣物，以防感冒。"

},

"cw":{

"txt":"较适宜洗车，未来一天无雨，风力较小，擦洗一新的汽车至少能保持一天。"

},

"sport":{

"txt":"天气较好，但考虑风力较强且气温较低，推荐您进行室内运动，若在户外运动注意防风并适当增减衣物。"

}

}

daily\_forecast的格式如下。

"daily\_forecast":[

{

"date":"2018-12-16",

"cond":{

"txt\_d":"多云"

},

"tmp":{

"max":"10",

"min":"1"

}

},

{

"date":"2018-12-17",

"cond":{

"txt\_d":"晴"

},

"tmp":{

"max":"12",

"min":"1"

}

},

...

]

### 3.1.3 获取数据

我们通过http来获取服务器的数据，这里我们选择了第三方开源库OkHttp来访问 http，OkHttp在接口封装和底层实现上均做了优化，是开发者常用网络通讯库。OkHttp的项目主页地址是：<https://github.com/square/okhttp>。

使用OkHttp我们只需要在app/build.gradle文件的dependencies闭包中添加如下内容，并重新编译即可使用。

dependencies {

implementation "com.squareup.okhttp3:okhttp:3.12.0"

}

这里我们定义一个静态方法来访问http，后面我们可以直接调用这个方法来根据不同的url，从服务器获取不同的数据。

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);

}

## 3.2 罗列全国所有省、市、县信息

### 3.2.1 数据库存储省、市、县

Android系统中内置了SQLite数据库，这大大简化了我们对于数据存储的复杂度，本程序也采用这一内置数据库。

Android在其官方提供的API中提供了SQLiteOpenHelper类来帮助开发者使用SQLite数据库。这个类虽然大大简化了数据库的操作，但是仍然不够简单，本程序采用第三发开源库——LitePal替代SQLiteOpenHelper。

LitePal采用了对象关系映射(ORM)模式，并将常用数据库功能进行了封装，使得开发者可以简单快捷的完成数据库操作。LitePal的项目主页为<http://github.com/LitePalFramework/LitePal>，其上有详细的使用文档。

⑴配置LitePal

配置LitePal库只需要在app/build.gradle文件的dependencies闭包中添加如下内容：

dependencies {

implementation 'org.litepal.android:java:3.0.0'

}

并重新编译，即可使用。（3.0.0为我使用时的最新版本，最新版本号可在上述GitHub项目主页上查看）。

LitePal采用xml文件的方式配置数据库。在mian文件夹下创建assets文件，并新建一个litepal.xml文件，编辑litepal.xml文件：

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

<litepal>

<dbname value="weather"></dbname>

<version value="1"></version>

<list>

<mapping class="com.example.weatherdemo.db.Province"></mapping>

<mapping class="com.example.weatherdemo.db.City"></mapping>

<mapping class="com.example.weatherdemo.db.County"></mapping>

</list>

</litepal>

其中，<dbname>标签用于指定数据库名，<version>用于指定数据库版本号，<list>用于指定所有的映射模型，即将实体类与数据库中的表联系起来，每个<mapping>表示一个表。

最后还有配置一下AndroidManifest.xml文件，为项目的application添加name属性：

<application

android:name="org.litepal.LitePalApplication"

…></application>

至此我们就完成了所有的基础配置。

⑵创建表格对应的实体类

我们已经在3.1.1中介绍了从服务器获取到的省、市、县的数据格式，这里我们只要创建对应的类继承自LitePalSupport，并将类的数据成员与表格的列相对应，并生成相应的get、set函数即可，LitePal会根据之前的xml配置自动将表格和类进行映射。本程序需要创建三个表格province，city和county，分别对应存储省、市和县三级数据。这里以province表为例，其他表格的创建与此类似，这里不再赘述。

public class Province extends LitePalSupport {

private int id;

private String provinceName;

private int provinceCode;

public int getId() {

return id;

}

public void setId(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;

}

}

⑶解析和处理服务器数据

现在我们已经准备好了本地数据库来存储数据，我们可以解析服务器的数据并将其存储到数据库中。同样省、市、县三级操作类似，所以我们这里以市级数据为例介绍解析和处理数据的方法。

public static boolean handleCityResponse(String response,int provinceId){

if(!TextUtils.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.setProvinceId(provinceId);

city.save();

}

return true;

}catch (JSONException e){

e.printStackTrace();

}

}

return false;

}

这里传入从服务器获取的数据response以及该市对应省的id。所有的服务器数据均是以JSON格式的方式存储，这里我们采用官方提供的JSONArray和JSONObject方法解析JSON数据。

我们可以看到使用LitePal操作数据库是很方便的，我们只需定义对应类的具体对象，每个对象对应该数据库表格的一行，并调用LitePal提供的save函数即可方便的完成插入一条数据的操作。

### 3.2.2 显示省、市、县

由于我们在后面还需要使用遍历省、市、县数据，所以这里我们将显示这三级数据的界面定义为一个fragment。在fragment中我们使用ListView来显示省市县三级数据。

fragment中的成员变量如下：

public static final int LEVEL\_PROVINCE=0;

public static final int LEVEL\_CITY=1;

public static final int LEVEL\_COUNTY=2;

private ProgressBar progressBar;

private TextView titleText;

private Button backButton;

private ListView listView;

private ArrayAdapter<String> adapter;

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

private List<Province> provinceList;//省列表

private List<City> cityList;//市列表

private List<County> countyList;//县列表

private Province selectedProvince;//现在被选的省份

private City selectedCity;//现在被选的城市

private int currentLevel;//现在选中的级别

在onCreateView函数中初始化控件并加载ListView的布局，这里直接采用系统中的simple\_list\_item\_1布局，具体代码不再给出。

在onActivityCreated函数中添加ListView的item点击事件，和返回按钮的点击事件。

public void onActivityCreated(@Nullable Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

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==LEVEL\_CITY){

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();

}

其中queryProvinces,queryCity和queryCounty分别为查询省、市、县的数据，这里我们仍然以queryCity为例进行说明。

private void queryCounties(){

titleText.setText(selectedCity.getCityName());

backButton.setVisibility(View.VISIBLE);

countyList=LitePal.where("cityid=?",String.valueOf(selectedCity.getId())).find(County.class);

if(countyList.size()>0){

dataList.clear();

for(County county:countyList){

dataList.add(county.getCountyName());

}

adapter.notifyDataSetChanged();

listView.setSelection(0);

currentLevel=LEVEL\_COUNTY;

}else {

int provinceCode=selectedProvince.getProvinceCode();

int cityCode=selectedCity.getCityCode();

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

queryFromServer(address,"county");

}

}

因为我们有本地数据库存储，所以我们先去数据库中查询。这里我们直接使用LitePal的where方法来条件查询数据库。如果本地数据库没有记录再通过调用queryFromServer方法从服务器中查询，并将其添加到本地数据库中。QueryFromServer根据传入的地址和类型调用3.1.3中的sendOkHttpRequest方法访问服务器数据。

private void queryFromServer(String address,final String type){

showProgressBar();

HttpUtil.sendOkHttpRequest(address, new Callback() {

@Override

public void onFailure(Call call, IOException e) {

getActivity().runOnUiThread(new Runnable() {

@Override

public void run() {

closeProgressBar();

Toast.makeText(getContext(), "加载失败", Toast.LENGTH\_SHORT).show();

}

});

}

@Override

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

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.getId());

}

if(result){

getActivity().runOnUiThread(new Runnable() {

@Override

public void run() {

closeProgressBar();

if("province".equals(type)){

queryProvinces();

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

queryCities();

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

queryCounties();

}

}

});

}

}

});

}

## 3.3 显示天气信息

### 3.3.1 解析天气数据

3.1.2中我们介绍过所有获取的天气信息的格式，在3.2.1中我们曾使用官方提供的JSONArray和JSONObject方法解析JSON数据，但是包含天气数据的JSON数据较为复杂，所以我们选择了第三方库GSON来解析天气信息。

GSON是谷歌公司提供的解析JSON数据的开源库，同之前的第三方开源库使用方法类似，这里我们只需要在app/build.gradle文件的dependencies闭包中添加如下内容，并重新编译即可。

dependencies {

implementation 'com.google.code.gson:gson:2.7'

}

类似于LitePal将数据库的表格和实体类相映射，GSON将JSON数据的对象和一个实体类相映射。例如一段JSON格式数据如下所示：{“name”:”Tom”,”age”:20}，那我们只需要定义一个Person类，并加入name和age，并调用fromJson(jsonData,Person.class)即可自动获得一个Person对象。因此这里我们只需要根据3.1.1中介绍的JSON数据格式来创建对应的实体类即可。

这里我们以basic为例，其他对象与之相似，这里不再赘述。一般情况下GSON要求实体类中成员变量的名字与对应JSON数据的键的名字相同，所以在Basic类中我们使用了@SerializedName注解的方式手动将JSON字段和Java字段之间建立映射，从而能够自己定义变量名。

public class Basic {

@SerializedName("city")

public String cityName;

@SerializedName("id")

public String weatherId;

public Update update;

public class Update{

@SerializedName("loc")

public String updateTime;

}

}

创建Weather类，引用所有的实体类。

public class Weather {

public String status;

public Basic basic;

public AQI aqi;

public Now now;

public Suggestion suggestion;

@SerializedName("daily\_forecast")

public List<Forecast> forecastList;

}

最后我们定义一个静态函数来处理服务器返回的JSON串。

public static Weather handleWeatherResponse(String response){

try{

JSONObject jsonObject=new JSONObject(response);

JSONArray jsonArray=jsonObject.getJSONArray("HeWeather");

String weatherContent=jsonArray.getJSONObject(0).toString();

return new Gson().fromJson(weatherContent,Weather.class);

}catch (Exception e){

e.printStackTrace();

}

return null;

}

### 3.3.2 显示天气数据

创建一个WeatherActivity活动，用于显示天气数据。

在onCreate函数中初始化所有的控件，并获取本地缓存，本程序中我们使用了SharedPreferences来作为本地缓存。如果有缓存则不需要去服务器中查询直接显示即可，否则则要去服务器上查找数据。

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_weather);

…

SharedPreferences preferences=PreferenceManager.getDefaultSharedPreferences(this);//本地数据存储

String weatherString=preferences.getString("weather",null);

if(weatherString!=null){

//有本地缓存时直接解析天气数据

Weather weather=Utility.handleWeatherResponse(weatherString);

mWeatherId=weather.basic.weatherId;

showWeatherInfo(weather);

}else{

//无缓存时去服务器查询天气

mWeatherId=getIntent().getStringExtra("weather\_id");

weatherLayout.setVisibility(View.VISIBLE);

requestWeather(mWeatherId);

}

}

requestWeather函数通过城市的id来获取城市对应的天气信息。

public void requestWeather(final String weatherId){

String weatherUrl="http://guolin.tech/api/weather?cityid="+weatherId+"&key=b16def293d3e439d91afc1e57ffa067d";

HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {

@Override

public void onFailure(Call call, IOException e) {

e.printStackTrace();

runOnUiThread(new Runnable() {

@Override

public void run() {

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

swipeRefresh.setRefreshing(false);

}

});

}

@Override

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

final String responseText=response.body().string();

final Weather weather=Utility.handleWeatherResponse(responseText);

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();

mWeatherId=weather.basic.weatherId;

showWeatherInfo(weather);

}else {

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

}

swipeRefresh.setRefreshing(false);

}

});

}

});

}

得到的数据存储在Weather对象中，再利用showWeatherInfo函数将对应数据在控件中显示出来就可，这里就不再展示代码。

在3.2.2中的选择省市县的onActivityCreated函数中添加一个if分支，即当用户选择了县之后跳转到天气界面。

else if (currentLevel==LEVEL\_COUNTY){

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

if(getActivity() instanceof MainActivity) {

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

intent.putExtra("weather\_id", weatherId);

startActivity(intent);

getActivity().finish();

}else if (getActivity() instanceof WeatherActivity){

WeatherActivity activity=(WeatherActivity)getActivity();

activity.drawerLayout.closeDrawers();

activity.swipeRefresh.setRefreshing(true);

activity.requestWeather(weatherId);

}

}

在MainActivity的onCreate中添加缓存数据是否存在的判断，这样再第二次启动程序后就可以直接查看之前选择的城市天气情况了。

SharedPreferences preferences=PreferenceManager.getDefaultSharedPreferences(this);

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

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

startActivity(intent);

finish();

}

## 3.4 下拉刷新

采用SwipeRefreshLayout实现下拉刷新的操作，我们只需要将需要支持下拉刷新的控件放在SwipeRefreshLayout中就可以让这个控件支持下拉刷新的功能。所以这里我们只需要将WeatherActivity中的控件放在SwipeRefreshLayout下。

<android.support.v4.widget.SwipeRefreshLayout

android:id="@+id/swipe\_refresh"

android:layout\_width="match\_parent"

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

<ScrollView

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:overScrollMode="never"

android:scrollbars="none">

…

</ScrollView>

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

并在onCreate方法中设置它的setOnRefreshListener方法就可以实现下拉刷新的功能。

swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

requestWeather(mWeatherId);

}

});

## 3.5 切换城市

采用DrawerLayout来实现从左侧屏幕边缘向右边滑动呼出选择城市的碎片界面。同时在左上角添加一个按钮让用户同时可以点击呼出。

修改activity\_weather.xml的布局。

<android.support.v4.widget.DrawerLayout

android:id="@+id/drawer\_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.SwipeRefreshLayout>

<fragment

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

android:name="com.example.weatherdemo.ChooseAreaFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_gravity="start"/>

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

并在onCreate中为按钮注册点击事件。

navButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

drawerLayout.openDrawer(GravityCompat.START);

}

});

同时修改选择省市县的碎片的onActivityCreate函数中currentLevel==LEVEL\_COUNTY分支，在这个分支中添加判断。通过getActivity()获取当前对应的Activity，如果当前在MainActivity中说明用户刚打开软件，则直接处理就可以，而如果当前在WeatherActivity中说明用户正通过DrawerLayout切换城市，则我们除了正常处理外，还需要关闭DrawerLayout。

## 3.6 自动更新

为了实现自动更新的功能，我们只需要创建一个在后天运行的定时任务，这里我们选择了Alarm，没有选择Timer类的原因是因为Android手机会在长时间不操作的情况下自动让CPU进入睡眠状态，这可能导致Timer中的任务无法正确运行。而Alarm则可以唤醒CPU。

创建一个Service类，它每隔8小时运行一次更新操作。

public class AutoUpdateService extends Service {

public AutoUpdateService() {

}

@Override

public IBinder onBind(Intent intent) {

// TODO: Return the communication channel to the service.

return null;

}

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

updateWeather();

AlarmManager manager=(AlarmManager)getSystemService(ALARM\_SERVICE);

int anHour=8\*60\*60\*1000;//8小时更新一次

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.ELAPSED\_REALTIME\_WAKEUP,triggerAtTime,pi);

return super.onStartCommand(intent, flags, startId);

}

private void updateWeather(){

SharedPreferences preferences=PreferenceManager.getDefaultSharedPreferences(this);

String weatherString=preferences.getString("weather",null);

if(weatherString!=null){

Weather weather=Utility.handleWeatherResponse(weatherString);

String weatherId=weather.basic.weatherId;

String weatherUrl="http://guolin.tech/api/weather?cityid="+weatherId+"&key=b16def293d3e439d91afc1e57ffa067d";

HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {

@Override

public void onFailure(Call call, IOException e) {

e.printStackTrace();

}

@Override

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

String responseText=response.body().string();

Weather weather=Utility.handleWeatherResponse(responseText);

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

SharedPreferences.Editor editor=PreferenceManager.getDefaultSharedPreferences(AutoUpdateService.this).edit();

editor.putString("weather",responseText);

editor.apply();

}

}

});

}

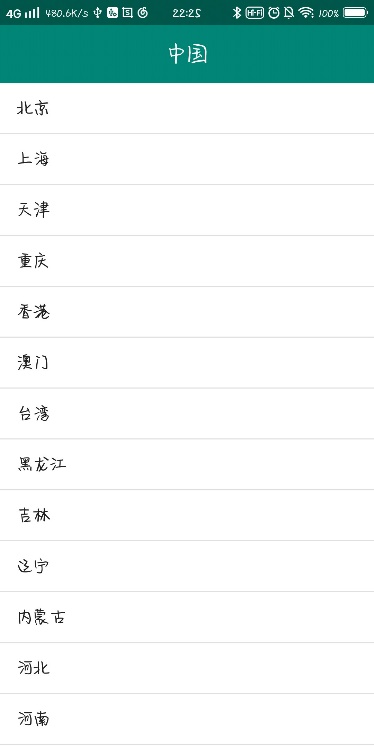
}

}

# 4 运行示例

程序第一次启动时需要选择城市，省、市、县的选择界面分别如图4.1(a)、(b)、(c)。

选择到县级之后就可以获取到具体的天气情况，如图4.2。

(a) (b) (c)

图4.1



图4.2

在该界面点击左上角home按钮或者从屏幕左侧往右滑动可以呼出城市切换界面，如图4.3。

在该界面下拉可以刷新当前天气信息，如图4.4。



图4.3



图4.4

# 5 收获与体会

这个程序是我第一次通过android平台开发一个程序，开发过程中遇到了很多问题，同时也借鉴了一些别人的程序。但是同时这也是一个学习的过程，通过完成这个程序的开发，我觉得我对于android的知识获得了极大的提高。这样一个程序的开发是对相关技术、自学能力等多方位能力的考查，很感谢这一次的开发经历，从中我不仅获得了技术的提高，更是综合能力的提高和自信的提升。