

第 14 章 进入实战，开发酷欧天气

我们将要在本章中编写一个功能较为完整的天气预报程序，学习了这么久的 Android 开发，现在终于到了考核验收的时候了。那么第一步我们需要给这个软件起个好听的名字，这里就叫它酷欧天气吧，英文名就叫做 Cool Weather。确定了名字之后，下面就可以开始动手了。

14.1 功能需求及技术可行性分析

在开始编码之前，我们需要先对程序进行需求分析，想一想酷欧天气中应该具备哪些功能。将这些功能全部整理出来之后，我们才好动手去一一实现。这里我认为酷欧天气中至少应该具备以下功能。

1. 可以罗列出全国所有的省、市、县。
2. 可以查看全国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。

虽然看上去只有四个主要的功能点，但如果想要全部实现这些功能却需要用到 UI、网络、定位、数据存储、服务等技术，因此还是非常考验你的综合应用能力的。不过好在这些技术在前面的章节中我们全部都学习过了，只要你学得用心，相信完成这些功能对你来说并不难。

分析完了需求之后，接下来就要进行技术可行性分析了。首先需要考虑的一个问题就是，我们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。很幸运，现在网上有不少免费的天气预报接口可以实现上述功能，如新浪天气、雅虎天气等，这里我们准备使用中国天气网提供的 API 接口来实现上述功能。

比如要想罗列出中国所有的省份，只需访问如下地址：

`http://www.weather.com.cn/data/list3/city.xml`

服务器会返回我们一段文本信息，其中包含了中国所有的省份名称以及省级代号，如下所示：

01|北京,02|上海,03|天津,04|重庆,05|黑龙江,06|吉林,07|辽宁,08|内蒙古,09|河北,10|山西,11|陕西,12|山东,13|新疆,14|西藏,15|青海,16|甘肃,17|宁夏,18|河南,19|江苏,20|湖北,21|浙江,22|安

徽,23|福建,24|江西,25|湖南,26|贵州,27|四川,28|广东,29|云南,30|广西,31|海南,32|香港,33|澳门,34|台湾

可以看到，北京的代号是 01，上海的代号是 02，不同省份之间以逗号分隔，省份名称和省级代号之间以单竖线分隔。那么如何才能知道某个省内有哪些城市呢？其实也很简单，比如江苏的省级代号是 19，访问如下地址即可：

```
http://www.weather.com.cn/data/list3/city19.xml
```

也就是说，只需要将省级代号添加到 city 的后面就行了，现在服务器返回的数据如下：

1901|南京,1902|无锡,1903|镇江,1904|苏州,1905|南通,1906|扬州,1907|盐城,1908|徐州,1909|淮安,1910|连云港,1911|常州,1912|泰州,1913|宿迁

这样我们就得到江苏省内所有城市的信息了，可以看到，现在返回的数据格式和刚才查看省份信息时返回的数据格式是一样的。相信此时你已经可以举一反三了，比如说苏州的市级代号是 1904，那么想要知道苏州市下又有哪些县的时候，只需访问如下地址：

```
http://www.weather.com.cn/data/list3/city1904.xml
```

这次服务器返回的数据如下：

190401|苏州,190402|常熟,190403|张家港,190404|昆山,190405|吴县东山,190406|吴县,190407|吴江,190408|太仓

通过这种方式，我们就能把全国所有的省、市、县都罗列出来了。那么解决了全国省市县数据的获取，我们又怎样才能查看到具体的天气信息呢？这就必须找到某个地区对应的天气代号。比如说昆山的县级代号是 190404，那么访问如下地址：

```
http://www.weather.com.cn/data/list3/city190404.xml
```

这时服务器返回的数据非常简短：

```
190404|101190404
```

其中，后半部分的 101190404 就是昆山所对应的天气代号了。这个时候再去访问查询天气接口，将相应的天气代号填入即可，接口地址如下：

```
http://www.weather.com.cn/data/cityinfo/101190404.html
```

这样，服务器就会把昆山当前的天气信息以 JSON 格式返回给我们了，如下所示：

```
{"weatherinfo":  
  {"city":"昆山","cityid":"101190404","temp1":"21℃","temp2":"9℃",  
   "weather":"多云转小雨","img1":"d1.gif","img2":"n7.gif","ptime":"11:00"}  
}
```

其中 city 表示城市名，cityid 表示城市对应的天气代号，temp1 和 temp2 表示气温是几度到几度，weather 表示今日天气信息的描述，img1 和 img2 表示今日天气对应的图片，ptime

表示天气发布的时间。至于 JSON 数据的解析，对你来说应该很轻松了吧。

确定了技术完全可行之后，接下来就可以开始编码了。不过别着急，我们准备让酷欧天气成为一个开源软件，并使用 GitHub 来进行代码托管，因此先让我们进入到本书最后一次的 Git 时间。

14.2 Git 时间，将代码托管到 GitHub 上

经过前面几章的学习，相信你已经可以非常熟练地使用 Git 了。本节依然是 Git 时间，这次我们将会把酷欧天气的代码托管到 GitHub 上面。

GitHub 是全球最大的代码托管网站，主要是借助 Git 来进行版本控制的。任何开源软件都可以免费地将代码提交到 GitHub 上，以零成本的代价进行代码托管。GitHub 的官网地址如下：

<https://github.com/>

官网的首页如图 14.1 所示。

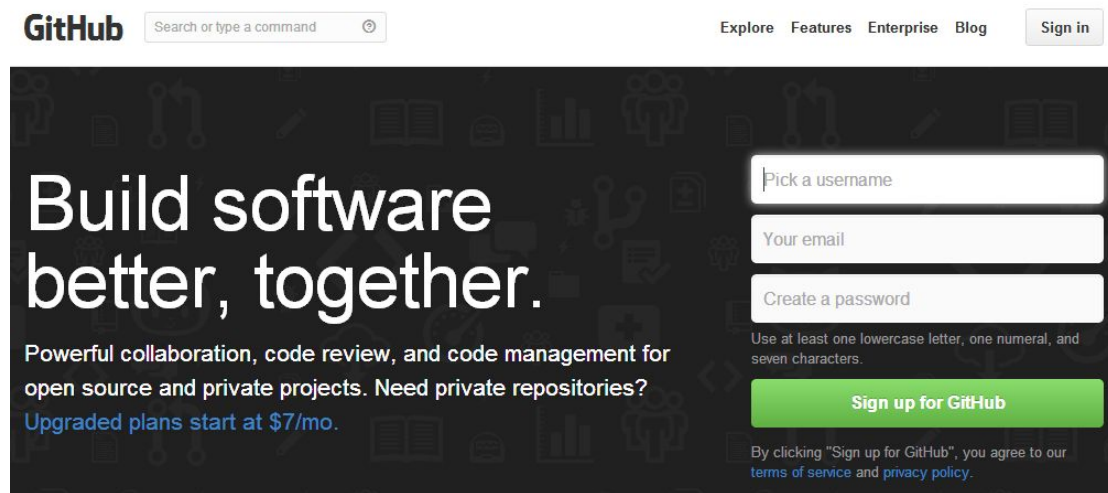
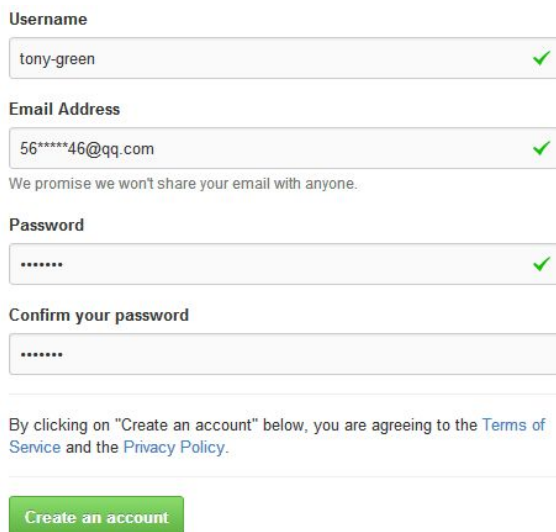


图 14.1

首先你需要有一个 GitHub 账号才能使用 GitHub 的代码托管功能，点击 Sign up for GitHub 按钮进行注册，然后填入用户名、邮箱和密码，如图 14.2 所示。



Registration form for GitHub account creation. The form includes fields for Username, Email Address, Password, and Confirm your password. Each field has a green checkmark indicating it is valid. Below the fields is a link to the Terms of Service and Privacy Policy, and a green button labeled 'Create an account'.

Username
tony-green ✓

Email Address
56*****46@qq.com ✓
We promise we won't share your email with anyone.

Password
***** ✓

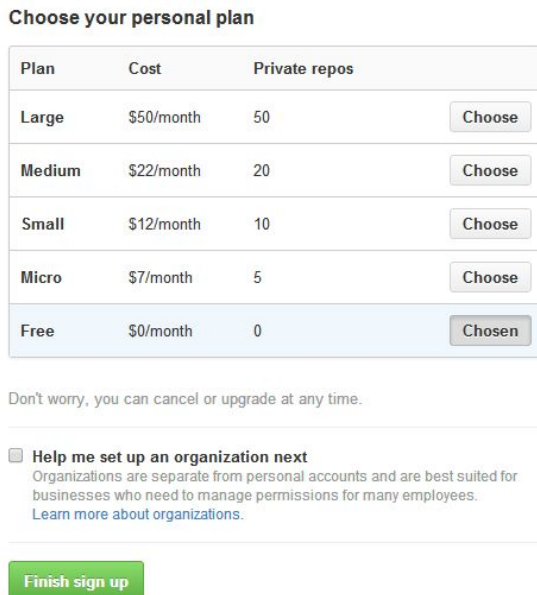
Confirm your password

By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

图 14.2

点击 Create an account 按钮来创建账户，接下来会让你选择个人计划，收费计划有创建私人版本库的权限，而我们的酷欧天气是开源软件，所以这里选择免费计划就可以了，如图 14.3 所示。



Choose your personal plan

Plan	Cost	Private repos	
Large	\$50/month	50	Choose
Medium	\$22/month	20	Choose
Small	\$12/month	10	Choose
Micro	\$7/month	5	Choose
Free	\$0/month	0	Chosen

Don't worry, you can cancel or upgrade at any time.

☐ Help me set up an organization next
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations.](#)

Finish sign up

图 14.3

接着点击 Finish sign up 按钮完成注册，就会跳转到 GitHub 的个人主界面了，如图 14.4 所示。

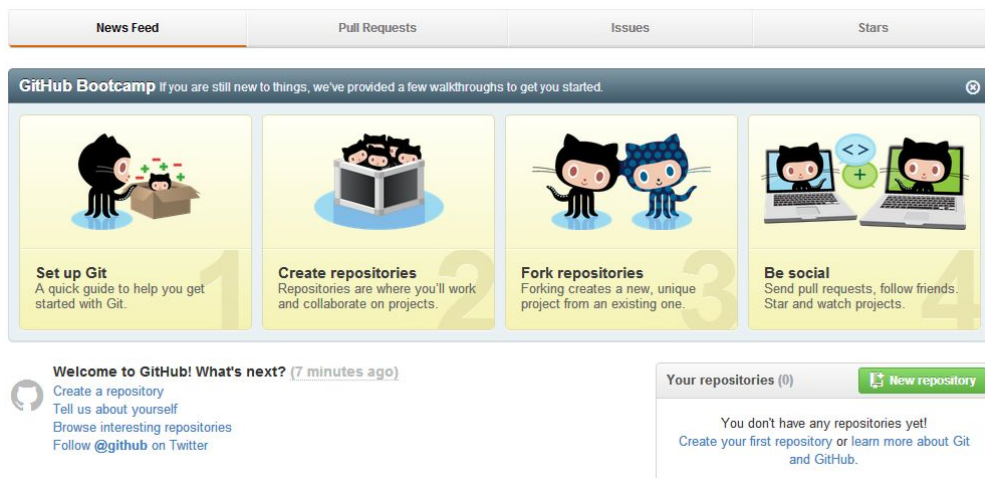


图 14.4

然后我们可以点击右下角的 New repository 按钮来创建一个版本库，这里将版本库命名为 coolweather，然后选择添加一个 Android 项目类型的 .gitignore 文件，并使用 Apache v2 License 来作为酷欧天气的开源协议，如图 14.5 所示。

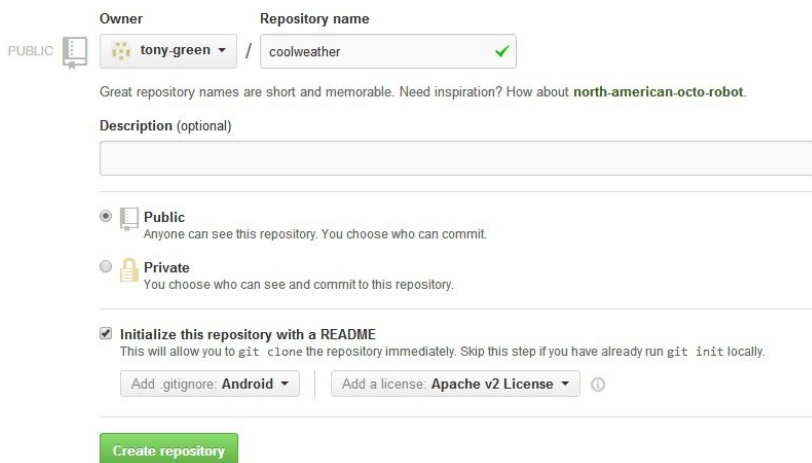


图 14.5

接着点击 Create repository 按钮，coolweather 这个版本库就创建完成了，如图 14.6 所示。版本库主页地址是 <https://github.com/tony-green/coolweather>。



图 14.6

可以看到，GitHub 已经自动帮我们创建了 .gitignore、LICENSE 和 README.md 这三个文件，其中编辑 README.md 文件中的内容可以修改酷欧天气版本库主页的描述。

创建好了版本库之后，我们就需要创建酷欧天气这个项目了。在 Eclipse 中新建一个 Android 项目，项目名叫做 CoolWeather，包名叫做 com.coolweather.app，仍然使用的是 4.0 的 API，如图 14.7 所示。

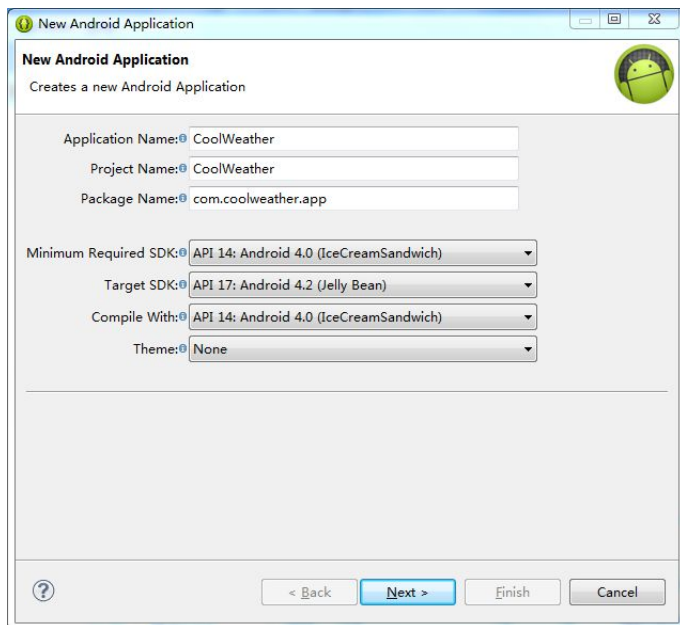


图 14.7

之后的步骤不用多说，一直点击 Next 就可以完成项目的创建，所有选项都使用默认的就好。

接下来的一步非常重要，我们需要将远程版本库克隆到本地。首先必须知道远程版本库的 Git 地址，可以在酷欧天气版本库主页的右下角找到，如图 14.8 所示。



图 14.8

点击右边的复制按钮可以将版本库的 Git 地址复制到剪贴板，酷欧天气版本库的 Git 地址是 `https://github.com/tony-green/coolweather.git`。然后打开 Git Bash 并切换到 CoolWeather 的工程目录下，如图 14.9 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine
$ cd f:

Tony@TONY-PC /f
$ cd codes/AndroidFirstLine/CoolWeather/

Tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather
$
```

图 14.9

接着输入 `git clone https://github.com/tony-green/coolweather.git` 来把远程版本库克隆到本地，如图 14.10 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather
$ git clone https://github.com/tony-green/coolweather.git
Cloning into 'coolweather'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 14 (delta 5), reused 0 (delta 0)
Unpacking objects: 100% (14/14), done.
Checking connectivity... done
```

图 14.10

看到图中所给的文字提示就表示克隆成功了，并且 `.gitignore`、`LICENSE` 和 `README.md` 这三个文件也已经被复制到了本地，可以进入到 `coolweather` 目录，并使用 `ls -al` 命令查看一

下，如图 14.11 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather
$ cd coolweather/

Tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather/coolweather (master)
$ ls -al
total 7
drwxr-xr-x  6 Tony   Administ  0 Feb  3 16:51 .
drwxr-xr-x 16 Tony   Administ  0 Feb  3 16:51 ..
drwxr-xr-x 12 Tony   Administ  0 Feb  3 16:51 .git
-rw-r--r--  1 Tony   Administ 361 Feb  3 16:51 .gitignore
-rw-r--r--  1 Tony   Administ 11527 Feb  3 16:51 LICENSE
-rw-r--r--  1 Tony   Administ 372 Feb  3 16:51 README.md
```

图 14.11

现在我们需要将这个目录中的所有文件全部复制到上一层目录中，这样就能将整个 CoolWeather 工程目录添加到版本控制中去了。注意.git 是一个隐藏目录，在复制的时候千万不要漏掉。复制完之后可以把 coolweather 目录删除掉，最终 CoolWeather 工程的目录结构如图 14.12 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather (master)
$ ls -al
total 35
drwxr-xr-x 19 Tony   Administ  0 Feb  3 17:01 .
drwxr-xr-x  1 Tony   Administ  0 Feb  3 16:32 ..
-rw-r--r--  1 Tony   Administ 364 Feb  3 16:32 .classpath
drwxr-xr-x 12 Tony   Administ  0 Feb  3 16:57 .git
-rw-r--r--  1 Tony   Administ 361 Feb  3 16:51 .gitignore
-rw-r--r--  1 Tony   Administ 847 Feb  3 16:32 .project
drwxr-xr-x  3 Tony   Administ  0 Feb  3 16:32 .settings
-rw-r--r--  1 Tony   Administ 879 Feb  3 16:32 AndroidManifest.xml
-rw-r--r--  1 Tony   Administ 11527 Feb  3 16:51 LICENSE
-rw-r--r--  1 Tony   Administ 372 Feb  3 16:51 README.md
drwxr-xr-x  2 Tony   Administ  0 Feb  3 16:32 assets
drwxr-xr-x  6 Tony   Administ  0 Feb  3 16:32 bin
drwxr-xr-x  3 Tony   Administ  0 Feb  3 16:32 gen
-rw-r--r--  1 Tony   Administ 51394 Feb  3 16:32 ic_launcher-web.png
drwxr-xr-x  3 Tony   Administ  0 Feb  3 16:32 libs
-rw-r--r--  1 Tony   Administ 781 Feb  3 16:32 proguard-project.txt
-rw-r--r--  1 Tony   Administ 563 Feb  3 16:32 project.properties
drwxr-xr-x 14 Tony   Administ  0 Feb  3 16:32 res
drwxr-xr-x  3 Tony   Administ  0 Feb  3 16:32 src
```

图 14.12

接下来我们应该把 CoolWeather 项目中现有的文件提交到 GitHub 上面，这就很简单了，先将所有文件添加到版本控制中，如下所示：

```
git add .
```


然后在本地执行提交操作：

```
git commit -m "First commit."
```

最后将提交的内容同步到远程版本库，也就是 GitHub 上面：


```
git push origin master
```

注意，最后一步的时候 GitHub 要求输入用户名和密码来进行身份校验，这里输入我们注册时填入的用户名和密码就可以了，如图 14.13 所示。

```
tony@TONY-PC /f/codes/AndroidFirstLine/CoolWeather <master>
$ git push origin master
Username for 'https://github.com': tony-green
Password for 'https://tony-green@github.com':
Counting objects: 41, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (26/26), done.
Writing objects: 100% (40/40), 431.82 KiB | 0 bytes/s, done.
Total 40 (delta 1), reused 0 (delta 0)
To https://github.com/tony-green/coolweather.git
7b5a385..929bbeb master -> master
```

图 14.13

这样就已经同步完成了，现在刷新一下酷欧天气版本库的主页，你会看到刚才提交的那些文件已经存在了，如图 14.14 所示。



branch: master coolweather		
First commit.		
tinuharo1971 authored 14 minutes ago latest commit 929bbebc7		
.settings	First commit.	14 minutes ago
libs	First commit.	14 minutes ago
res	First commit.	14 minutes ago
src	First commit.	14 minutes ago
.gitignore	Initial commit	8 hours ago
AndroidManifest.xml	First commit.	14 minutes ago
LICENSE	Initial commit	8 hours ago
README.md	Update README.md	6 hours ago
ic_launcher-web.png	First commit.	14 minutes ago
proguard-project.txt	First commit.	14 minutes ago
project.properties	First commit.	14 minutes ago

图 14.14

14.3 创建数据库和表

从本节开始，我们就要真正地动手编码了，为了要让项目能够有更好的结构，这里需要在 `com.coolweather.app` 包下再新建几个包，如图 14.15 所示。

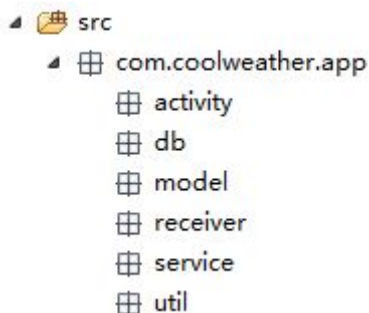


图 14.15

其中 `activity` 包用于存放所有活动相关的代码，`db` 包用于存放所有数据库相关的代码，`model` 包用于存放所有模型相关的代码，`receiver` 包用于存放所有广播接收器相关的代码，`service` 包用于存放所有服务相关的代码，`util` 包用于存放所有工具相关的代码。ADT 帮我们自动生成的 `MainActivity` 和 `activity_main.xml` 文件就不需要了，这里直接将它们删除掉。

根据 14.1 节进行的技术可行性分析，首先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。表的设计当然是仁者见仁智者见智，并不是说哪种设计就是最规范最完美的。这里我准备建立三张表，`Province`、`City`、`County`，分别用于存放省、市、县的各种数据信息，三张表的建表语句分别如下。

Province:

```
create table Province (
    id integer primary key autoincrement,
    province_name text,
    province_code text)
```

其中 `id` 是自增长主键，`province_name` 表示省名，`province_code` 表示省级代号。

City:

```
create table City (
    id integer primary key autoincrement,
    city_name text,
    city_code text,
    province_id integer)
```

其中 id 是自增长主键，city_name 表示城市名，city_code 表示市级代号，province_id 是 City 表关联 Province 表的外键。

```
County:
create table County (
    id integer primary key autoincrement,
    county_name text,
    county_code text,
    city_id integer)
```

其中 id 是自增长主键，county_name 表示县名，county_code 表示县级代号，city_id 是 County 表关联 City 表的外键。

建表语句就是这样，接下来我们将建表语句写入到代码中。在 db 包下新建一个 CoolWeatherOpenHelper 类，代码如下所示：

```
public class CoolWeatherOpenHelper extends SQLiteOpenHelper {

    /**
     * Province 建表语句
     */
    public static final String CREATE_PROVINCE = "create table Province ("
        + "id integer primary key autoincrement, "
        + "province_name text, "
        + "province_code text)";

    /**
     * City 建表语句
     */
    public static final String CREATE_CITY = "create table City ("
        + "id integer primary key autoincrement, "
        + "city_name text, "
        + "city_code text, "
        + "province_id integer)";

    /**
     * County 建表语句
     */
    public static final String CREATE_COUNTY = "create table County ("
        + "id integer primary key autoincrement, "
        + "county_name text, "
        + "county_code text, "
        + "city_id integer)";
```

```
public CoolWeatherOpenHelper(Context context, String name, CursorFactory
factory, int version) {
    super(context, name, factory, version);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_PROVINCE); // 创建Province表
    db.execSQL(CREATE_CITY); // 创建City表
    db.execSQL(CREATE_COUNTY); // 创建County表
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
}

}
```

上面的代码非常好理解，将三条建表语句定义成常量，然后在 onCreate()方法中去执行创建就可以了。

另外，每张表在代码中最好能有一个对应的实体类，这样会非常便于我们后续的开发工作。因此，在 model 包下新建一个 Province 类，代码如下所示：

```
public class Province {
    private int id;
    private String provinceName;
    private String 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 String getProvinceCode() {
        return provinceCode;
    }

    public void setProvinceCode(String provinceCode) {
        this.provinceCode = provinceCode;
    }
}

```

接着在 model 包下新建一个 City 类，代码如下所示：

```

public class City {
    private int id;
    private String cityName;
    private String cityCode;
    private int provinceId;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCityName() {
        return cityName;
    }

    public void setCityName(String cityName) {
        this.cityName = cityName;
    }

    public String getCityCode() {
        return cityCode;
    }
}

```

```
public void setCityCode(String cityCode) {
    this.cityCode = cityCode;
}

public int getProvinceId() {
    return provinceId;
}

public void setProvinceId(int provinceId) {
    this.provinceId = provinceId;
}
}
```

然后在 model 包下新建一个 County 类，代码如下所示：

```
public class County {
    private int id;
    private String countyName;
    private String countyCode;
    private int cityId;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCountyName() {
        return countyName;
    }

    public void setCountyName(String countyName) {
        this.countyName = countyName;
    }

    public String getCountyCode() {
        return countyCode;
    }
}
```

```

    public void setCountyCode(String countyCode) {
        this.countyCode = countyCode;
    }

    public int getCityId() {
        return cityId;
    }

    public void setCityId(int cityId) {
        this.cityId = cityId;
    }
}

```

可以看到，实体类的内容都非常简单，基本就是生成数据库表对应字段的 get 和 set 方法就可以了。接着我们还需要创建一个 CoolWeatherDB 类，这个类将会把一些常用的数据库操作封装起来，以方便我们后面使用，代码如下所示：

```

public class CoolWeatherDB {

    /**
     * 数据库名
     */
    public static final String DB_NAME = "cool_weather";

    /**
     * 数据库版本
     */
    public static final int VERSION = 1;

    private static CoolWeatherDB coolWeatherDB;

    private SQLiteDatabase db;

    /**
     * 将构造方法私有化
     */
    private CoolWeatherDB(Context context) {
        CoolWeatherOpenHelper dbHelper = new CoolWeatherOpenHelper(context,
            DB_NAME, null, VERSION);
    }
}

```

```
        db = dbHelper.getWritableDatabase();
    }

    /**
     * 获取CoolWeatherDB的实例。
     */
    public synchronized static CoolWeatherDB getInstance(Context context) {
        if (coolWeatherDB == null) {
            coolWeatherDB = new CoolWeatherDB(context);
        }
        return coolWeatherDB;
    }

    /**
     * 将Province实例存储到数据库。
     */
    public void saveProvince(Province province) {
        if (province != null) {
            ContentValues values = new ContentValues();
            values.put("province_name", province.getProvinceName());
            values.put("province_code", province.getProvinceCode());
            db.insert("Province", null, values);
        }
    }

    /**
     * 从数据库读取全国所有的省份信息。
     */
    public List<Province> loadProvinces() {
        List<Province> list = new ArrayList<Province>();
        Cursor cursor = db
            .query("Province", null, null, null, null, null, null);
        if (cursor.moveToFirst()) {
            do {
                Province province = new Province();
                province.setId(cursor.getInt(cursor.getColumnIndex("id")));
                province.setProvinceName(cursor.getString(cursor
                    .getColumnIndex("province_name")));
                province.setProvinceCode(cursor.getString(cursor
```



```

        .getColumnIndex("province_code"))));
        list.add(province);
    } while (cursor.moveToNext());
}
return list;
}

/**
 * 将City实例存储到数据库。
 */
public void saveCity(City city) {
    if (city != null) {
        ContentValues values = new ContentValues();
        values.put("city_name", city.getCityName());
        values.put("city_code", city.getCityCode());
        values.put("province_id", city.getProvinceId());
        db.insert("City", null, values);
    }
}

/**
 * 从数据库读取某省下所有的城市信息。
 */
public List<City> loadCities(int provinceId) {
    List<City> list = new ArrayList<City>();
    Cursor cursor = db.query("City", null, "province_id = ?",
        new String[] { String.valueOf(provinceId) }, null, null, null);
    if (cursor.moveToFirst()) {
        do {
            City city = new City();
            city.setId(cursor.getInt(cursor.getColumnIndex("id")));
            city.setCityName(cursor.getString(cursor
                .getColumnIndex("city_name")));
            city.setCityCode(cursor.getString(cursor
                .getColumnIndex("city_code")));
            city.setProvinceId(provinceId);
            list.add(city);
        } while (cursor.moveToNext());
    }
}

```

```
        return list;
    }

    /**
     * 将County实例存储到数据库。
     */
    public void saveCounty(County county) {
        if (county != null) {
            ContentValues values = new ContentValues();
            values.put("county_name", county.getCountyName());
            values.put("county_code", county.getCountyCode());
            values.put("city_id", county.getCityId());
            db.insert("County", null, values);
        }
    }

    /**
     * 从数据库读取某城市下所有的县信息。
     */
    public List<County> loadCounties(int cityId) {
        List<County> list = new ArrayList<County>();
        Cursor cursor = db.query("County", null, "city_id = ?",
            new String[] { String.valueOf(cityId) }, null, null, null);
        if (cursor.moveToFirst()) {
            do {
                County county = new County();
                county.setId(cursor.getInt(cursor.getColumnIndex("id")));
                county.setCountyName(cursor.getString(cursor
                    .getColumnIndex("county_name")));
                county.setCountyCode(cursor.getString(cursor
                    .getColumnIndex("county_code")));
                county.setCityId(cityId);
                list.add(county);
            } while (cursor.moveToNext());
        }
        return list;
    }
}
```

可以看到，CoolWeatherDB 是一个单例类，我们将它的构造方法私有化，并提供了一个 getInstance() 方法来获取 CoolWeatherDB 的实例，这样就可以保证全局范围内只会有一个 CoolWeatherDB 的实例。接下来我们在 CoolWeatherDB 中提供了六组方法，saveProvince()、loadProvinces()、saveCity()、loadCities()、saveCounty()、loadCounties()，分别用于存储省份数据、读取所有省份数据、存储城市数据、读取某省内所有城市数据、存储县数据、读取某市内所有县的数据。有了这几个方法，我们后面很多的数据库操作都将会变得非常简单。

好了，第一阶段的代码写到这里就差不多了，我们现在提交一下。首先将所有新增的文件添加到版本控制中：

```
git add .
```

然后把 MainActivity 和 activity_main.xml 这两个文件从版本控制中删除掉：

```
git rm src/com/coolweather/app/MainActivity.java
```

```
git rm res/layout/activity_main.xml
```

接着执行提交操作：

```
git commit -m "新增数据库帮助类，以及各表对应的实体类。"
```

最后将提交同步到 GitHub 上面：

```
git push origin master
```

OK！第一阶段完工，下面让我们赶快进入到第二阶段的开发工作中吧。

14.4 遍历全国省市县数据

在第二阶段中，我们准备把遍历全国省市县的功能加入，这一阶段需要编写的代码量比较大，你一定要跟上脚步。

我们已经知道，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，所以我们可以在 util 包下先增加一个 HttpUtil 类，代码如下所示：

```
public class HttpUtil {

    public static void sendHttpRequest(final String address,
                                       final HttpCallbackListener listener) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                try {
                    URL url = new URL(address);
```

```
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setConnectTimeout(8000);
        connection.setReadTimeout(8000);
        InputStream in = connection.getInputStream();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
        StringBuilder response = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            response.append(line);
        }
        if (listener != null) {
            // 回调onFinish()方法
            listener.onFinish(response.toString());
        }
    } catch (Exception e) {
        if (listener != null) {
            // 回调onError()方法
            listener.onError(e);
        }
    } finally {
        if (connection != null) {
            connection.disconnect();
        }
    }
}

}).start();
}

}
```

HttpUtil 类中使用到了 `HttpCallbackListener` 接口来回调服务返回的结果，因此我们还需要在 `util` 包下添加这个接口，如下所示：

```
public interface HttpCallbackListener {
    void onFinish(String response);

    void onError(Exception e);
}
```

另外，由于服务器返回的省市县数据都是“代号|城市,代号|城市”这种格式的，所以我们最好再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 synchronized static boolean handleProvincesResponse(CoolWeatherDB
coolWeatherDB, String response) {
        if (!TextUtils.isEmpty(response)) {
            String[] allProvinces = response.split(",");
            if (allProvinces != null && allProvinces.length > 0) {
                for (String p : allProvinces) {
                    String[] array = p.split("\\|");
                    Province province = new Province();
                    province.setProvinceCode(array[0]);
                    province.setProvinceName(array[1]);
                    // 将解析出来的数据存储到Province表
                    coolWeatherDB.saveProvince(province);
                }
                return true;
            }
        }
        return false;
    }

    /**
     * 解析和处理服务器返回的市级数据
     */
    public static boolean handleCitiesResponse(CoolWeatherDB coolWeatherDB,
        String response, int provinceId) {
        if (!TextUtils.isEmpty(response)) {
            String[] allCities = response.split(",");
            if (allCities != null && allCities.length > 0) {
                for (String c : allCities) {
                    String[] array = c.split("\\|");
                    City city = new City();
```

```

        city.setCityCode(array[0]);
        city.setCityName(array[1]);
        city.setProvinceId(provinceId);
        // 将解析出来的数据存储到City表
        coolWeatherDB.saveCity(city);
    }
    return true;
}
}
return false;
}

/**
 * 解析和处理服务器返回的县级数据
 */
public static boolean handleCountiesResponse(CoolWeatherDB coolWeatherDB,
        String response, int cityId) {
    if (!TextUtils.isEmpty(response)) {
        String[] allCounties = response.split(",");
        if (allCounties != null && allCounties.length > 0) {
            for (String c : allCounties) {
                String[] array = c.split("\\|");
                County county = new County();
                county.setCountyCode(array[0]);
                county.setCountyName(array[1]);
                county.setCityId(cityId);
                // 将解析出来的数据存储到County表
                coolWeatherDB.saveCounty(county);
            }
            return true;
        }
    }
    return false;
}
}
}

```

可以看到，我们提供了 `handleProvincesResponse()`、`handleCitiesResponse()`、`handleCountiesResponse()` 这三个方法，分别用于解析和处理服务器返回的省级、市级和县级数据。解析的规则就是先按逗号分隔，再按单竖线分隔，接着将解析出来的数据设置到实体

类中，最后调用 CoolWeatherDB 中的三个 save() 方法将数据存储到相应的表中。

需要准备的工具类就这么多，现在我们可以开始写界面了。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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="#484E61" >

        <TextView
            android:id="@+id/title_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textColor="#fff"
            android:textSize="24sp" />
    </RelativeLayout>

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```

布局文件中的内容比较简单，我们先是定义了一个 50dp 高的头布局，并在里面放置了一个 TextView 用于显示标题内容。然后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。

接下来也是最关键的一步，我们需要编写用于遍历省市县数据的活动了。在 activity 包下新建 ChooseAreaActivity 继承自 Activity，代码如下所示：

```
public class ChooseAreaActivity extends Activity {

    public static final int LEVEL_PROVINCE = 0;
```

```
public static final int LEVEL_CITY = 1;
public static final int LEVEL_COUNTY = 2;

private ProgressDialog progressDialog;
private TextView titleText;
private ListView listView;
private ArrayAdapter<String> adapter;
private CoolWeatherDB coolWeatherDB;
private List<String> dataList = new ArrayList<String>();
/**
 * 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;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.choose_area);
    listView = (ListView) findViewById(R.id.list_view);
```



```

        titleText = (TextView) findViewById(R.id.title_text);
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_
list_item_1, dataList);
        listView.setAdapter(adapter);
        coolWeatherDB = CoolWeatherDB.getInstance(this);
        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View view, int index,
                long arg3) {
                if (currentLevel == LEVEL_PROVINCE) {
                    selectedProvince = provinceList.get(index);
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    selectedCity = cityList.get(index);
                    queryCounties();
                }
            }
        });
        queryProvinces(); // 加载省级数据
    }

/**
 * 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() {
    provinceList = coolWeatherDB.loadProvinces();
    if (provinceList.size() > 0) {
        dataList.clear();
        for (Province province : provinceList) {
            dataList.add(province.getProvinceName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        titleText.setText("中国");
        currentLevel = LEVEL_PROVINCE;
    } else {
        queryFromServer(null, "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() {
    cityList = coolWeatherDB.loadCities(selectedProvince.getId());
    if (cityList.size() > 0) {
        dataList.clear();
        for (City city : cityList) {
            dataList.add(city.getCityName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        titleText.setText(selectedProvince.getProvinceName());
        currentLevel = LEVEL_CITY;
    } else {
        queryFromServer(selectedProvince.getProvinceCode(), "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() {
    countyList = coolWeatherDB.loadCounties(selectedCity.getId());
    if (countyList.size() > 0) {
        dataList.clear();
        for (County county : countyList) {
            dataList.add(county.getCountyName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        titleText.setText(selectedCity.getCityName());
        currentLevel = LEVEL_COUNTY;
    } else {
        queryFromServer(selectedCity.getCityCode(), "county");
    }
}

/**
 * 根据传入的代号和类型从服务器上查询省市县数据。
 */
```

```

    */
    private void queryFromServer(final String code, final String type) {
        String address;
        if (!TextUtils.isEmpty(code)) {
            address = "http://www.weather.com.cn/data/list3/city" + code +
".xml";
        } else {
            address = "http://www.weather.com.cn/data/list3/city.xml";
        }
        showProgressDialog();
        HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
            @Override
            public void onFinish(String response) {
                boolean result = false;
                if ("province".equals(type)) {
                    result = Utility.handleProvincesResponse(coolWeatherDB,
                        response);
                } else if ("city".equals(type)) {
                    result = Utility.handleCitiesResponse(coolWeatherDB,
                        response, selectedProvince.getId());
                } else if ("county".equals(type)) {
                    result = Utility.handleCountiesResponse(coolWeatherDB,
                        response, selectedCity.getId());
                }
                if (result) {
                    // 通过runOnUiThread()方法回到主线程处理逻辑
                    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();
                            }
                        }
                    });
                }
            }
        });
    }

```

```
    }

    @Override
    public void onError(Exception e) {
        // 通过runOnUiThread()方法回到主线程处理逻辑
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                closeProgressDialog();
                Toast.makeText(ChooseAreaActivity.this,
                    "加载失败", Toast.LENGTH_SHORT).show();
            }
        });
    }
}

/**
 * 显示进度对话框
 */
private void showProgressDialog() {
    if (progressDialog == null) {
        progressDialog = new ProgressDialog(this);
        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();
    }
}

/**
 * 捕获Back按键，根据当前的级别来判断，此时应该返回市列表、省列表、还是直接退出。
```

```

        */
        @Override
        public void onBackPressed() {
            if (currentLevel == LEVEL_COUNTY) {
                queryCities();
            } else if (currentLevel == LEVEL_CITY) {
                queryProvinces();
            } else {
                finish();
            }
        }
    }
}

```

这个类里的代码虽然非常多，可是逻辑却不复杂，我们来慢慢理一下。在 `onCreate()` 方法中先是获取到了一些控件的实例，然后去初始化了 `ArrayAdapter`，将它设置为 `ListView` 的适配器。之后又去获取到了 `CoolWeatherDB` 的实例，并给 `ListView` 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 `onCreate()` 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()` 方法的内部会首先调用 `CoolWeatherDB` 的 `loadProvinces()` 方法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，如果没有读取到就调用 `queryFromServer()` 方法来从服务器上查询数据。

`queryFromServer()` 方法会先根据传入的参数来拼装查询地址，这个地址就是我们在 14.1 节分析过的。确定了查询地址之后，接下来就调用 `HttpUtil` 的 `sendHttpRequest()` 方法来向服务器发送请求，响应的数据会回调到 `onFinish()` 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。接下来的一步很关键，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 `runOnUiThread()` 方法来实现从子线程切换到主线程，它的实现原理其实也是基于异步消息处理机制的。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。

当你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，这里就不重复讲解了。

另外还有一点需要注意，我们重写了 `onBackPressed()` 方法来覆盖默认 `Back` 键的行为，这里会根据当前的级别来判断是返回市级列表、省级列表、还是直接退出。

这样就把所有逻辑都梳理了一遍，你是不是觉得清晰多了？现在第二阶段的开发工作也完成得差不多了，我们可以运行一下来看看效果。不过在运行之前还有一件事没有做，当然就是配置 AndroidManifest.xml 文件了。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.app"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.coolweather.app.activity.ChooseAreaActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

主要就修改了两点，第一是添加了访问网络的权限，第二是将 ChooseAreaActivity 配置成主活动，这样一旦打开程序就会直接进入 ChooseAreaActivity 了。

现在可以运行一下程序了，结果如图 14.16 所示。



图 14.16

可以看到，全国所有省级单位都显示出来了，我们还可以继续查看市级单位，比如点击山东省，结果如图 14.17 所示。



图 14.17

然后再点击青岛市，结果如图 14.18 所示。



图 14.18

好了，这样第二阶段的开发工作也都完成了，我们仍然要把代码提交一下。

```
git add .  
git commit -m "完成遍历省市县三级列表的功能。"  
git push origin master
```

到目前为止进度算是相当不错啊，那么我们就趁热打铁，来进行第三阶段的开发工作。

14.5 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。天气信息应该在一个新的界面进行展示，因此这里需要创建一个新的活动和布局文件。

首先来创建布局文件吧，我们应该先思考布局文件中需要放置哪些控件，这就要由服务器返回的天气数据来决定了，如下所示：

```
{"weatherinfo":  
  {"city": "昆山", "cityid": "101190404", "temp1": "21℃", "temp2": "9℃",  
   "weather": "多云转小雨", "img1": "d1.gif", "img2": "n7.gif", "ptime": "11:00"}  
}
```


总共就这么多，其中 cityid 是用户无需知晓的，img1 和 img2 我们不准备使用，因此这里能够显示在界面上的就只有城市名、温度范围、天气信息描述、发布时间这几项，接下来就是合理地安排这几项数据的显示位置了。在 res/layout 目录中新建 weather_layout.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="#484E61" >

        <TextView
            android:id="@+id/city_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textColor="#fff"
            android:textSize="24sp" />
    </RelativeLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="#27A5F9" >

        <TextView
            android:id="@+id/publish_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_marginRight="10dp"
            android:layout_marginTop="10dp"
            android:textColor="#FFF"
            android:textSize="18sp" />
    </RelativeLayout>
</LinearLayout>
```

```
<LinearLayout
    android:id="@+id/weather_info_layout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:orientation="vertical" >

<TextView
    android:id="@+id/current_date"
    android:layout_width="wrap_content"
    android:layout_height="40dp"
    android:gravity="center"
    android:textColor="#FFF"
    android:textSize="18sp" />

<TextView
    android:id="@+id/weather_desp"
    android:layout_width="wrap_content"
    android:layout_height="60dp"
    android:layout_gravity="center_horizontal"
    android:gravity="center"
    android:textColor="#FFF"
    android:textSize="40sp" />

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="60dp"
    android:layout_gravity="center_horizontal"
    android:orientation="horizontal" >

<TextView
    android:id="@+id/temp1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:textColor="#FFF"
    android:textSize="40sp" />

<TextView
    android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:text="~"
        android:textColor="#FFF"
        android:textSize="40sp" />

        <TextView
            android:id="@+id/temp2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textColor="#FFF"
            android:textSize="40sp" />
    </LinearLayout>
</LinearLayout>
</RelativeLayout>
</LinearLayout>

```

在这个布局文件中，我们并没有用到任何特殊的控件，基本就是使用 `TextView` 来显示数据信息，然后嵌套多层 `LinearLayout` 和 `RelativeLayout` 来控制 `TextView` 的显示位置，相信你仔细分析一下就明白了，这里不再进行解释。

然后我们还需要在 `Utility` 类中添加几个方法，用于解析和处理服务返回的 JSON 数据，如下所示：

```

public class Utility {
    .....

    /**
     * 解析服务器返回的JSON数据，并将解析出的数据存储到本地。
     */
    public static void handleWeatherResponse(Context context, String response) {
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONObject weatherInfo = jsonObject.getJSONObject("weatherinfo");
            String cityName = weatherInfo.getString("city");
            String weatherCode = weatherInfo.getString("cityid");
            String temp1 = weatherInfo.getString("temp1");
            String temp2 = weatherInfo.getString("temp2");
            String weatherDesp = weatherInfo.getString("weather");

```

```

        String publishTime = weatherInfo.getString("ptime");
        saveWeatherInfo(context, cityName, weatherCode, temp1, temp2,
weatherDesp, publishTime);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

/**
 * 将服务器返回的所有天气信息存储到SharedPreferences文件中。
 */
public static void saveWeatherInfo(Context context, String cityName,
String weatherCode, String temp1, String temp2, String weatherDesp, String
publishTime) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy年M月d日",
Locale.CHINA);
    SharedPreferences.Editor editor = PreferenceManager
.getDefaultSharedPreferences(context).edit();
    editor.putBoolean("city_selected", true);
    editor.putString("city_name", cityName);
    editor.putString("weather_code", weatherCode);
    editor.putString("temp1", temp1);
    editor.putString("temp2", temp2);
    editor.putString("weather_desp", weatherDesp);
    editor.putString("publish_time", publishTime);
    editor.putString("current_date", sdf.format(new Date()));
    editor.commit();
}
}

```

其中 `handleWeatherResponse()` 方法用于将 JSON 格式的天气信息全部解析出来，`saveWeatherInfo()` 方法用于将这些数据都存储到 `SharedPreferences` 文件中。

接下来应该创建活动了，在 `activity` 包下新建 `WeatherActivity` 继承自 `Activity`，代码如下所示：

```

public class WeatherActivity extends Activity implements OnClickListener{

    private LinearLayout weatherInfoLayout;
    /**
     * 用于显示城市名

```

```

        */
private TextView cityNameText;
/**
 * 用于显示发布时间
 */
private TextView publishText;
/**
 * 用于显示天气描述信息
 */
private TextView weatherDespText;
/**
 * 用于显示气温1
 */
private TextView temp1Text;
/**
 * 用于显示气温2
 */
private TextView temp2Text;
/**
 * 用于显示当前日期
 */
private TextView currentDateText;
/**
 * 切换城市按钮
 */
private Button switchCity;
/**
 * 更新天气按钮
 */
private Button refreshWeather;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.weather_layout);
    // 初始化各控件
    weatherInfoLayout = (LinearLayout) findViewById(R.id.weather_
info_layout);
    cityNameText = (TextView) findViewById(R.id.city_name);

```

```
publishText = (TextView) findViewById(R.id.publish_text);
weatherDespText = (TextView) findViewById(R.id.weather_desp);
temp1Text = (TextView) findViewById(R.id.temp1);
temp2Text = (TextView) findViewById(R.id.temp2);
currentDateText = (TextView) findViewById(R.id.current_date);
switchCity = (Button) findViewById(R.id.switch_city);
refreshWeather = (Button) findViewById(R.id.refresh_weather);
String countyCode = getIntent().getStringExtra("county_code");
if (!TextUtils.isEmpty(countyCode)) {
    // 有县级代号时就去查询天气
    publishText.setText("同步中...");
    weatherInfoLayout.setVisibility(View.INVISIBLE);
    cityNameText.setVisibility(View.INVISIBLE);
    queryWeatherCode(countyCode);
} else {
    // 没有县级代号时就直接显示本地天气
    showWeather();
}
switchCity.setOnClickListener(this);
refreshWeather.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.switch_city:
            Intent intent = new Intent(this, ChooseAreaActivity.class);
            intent.putExtra("from_weather_activity", true);
            startActivity(intent);
            finish();
            break;
        case R.id.refresh_weather:
            publishText.setText("同步中...");
            SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
            String weatherCode = prefs.getString("weather_code", "");
            if (!TextUtils.isEmpty(weatherCode)) {
                queryWeatherInfo(weatherCode);
            }
            break;
    }
}
```

```

        default:
            break;
    }
}

/**
 * 查询县级代号所对应的天气代号。
 */
private void queryWeatherCode(String countyCode) {
    String address = "http://www.weather.com.cn/data/list3/city" +
countyCode + ".xml";
    queryFromServer(address, "countyCode");
}

/**
 * 查询天气代号所对应的天气。
 */
private void queryWeatherInfo(String weatherCode) {
    String address = "http://www.weather.com.cn/data/cityinfo/" +
weatherCode + ".html";
    queryFromServer(address, "weatherCode");
}

/**
 * 根据传入的地址和类型去向服务器查询天气代号或者天气信息。
 */
private void queryFromServer(final String address, final String type) {
    HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
        @Override
        public void onFinish(final String response) {
            if ("countyCode".equals(type)) {
                if (!TextUtils.isEmpty(response)) {
                    // 从服务器返回的数据中解析出天气代号
                    String[] array = response.split("\\|");
                    if (array != null && array.length == 2) {
                        String weatherCode = array[1];
                        queryWeatherInfo(weatherCode);
                    }
                }
            } else if ("weatherCode".equals(type)) {

```

```
// 处理服务器返回的天气信息
Utility.handleWeatherResponse(WeatherActivity.this,
response);

    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            showWeather();
        }
    });
}

@Override
public void onError(Exception e) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            publishText.setText("同步失败");
        }
    });
}

});
}

/**
 * 从SharedPreferences文件中读取存储的天气信息，并显示到界面上。
 */
private void showWeather() {
    SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
    cityNameText.setText( prefs.getString("city_name", ""));
    temp1Text.setText(prefs.getString("temp1", ""));
    temp2Text.setText(prefs.getString("temp2", ""));
    weatherDespText.setText(prefs.getString("weather_desp", ""));
    publishText.setText("今天" + prefs.getString("publish_time", "") + "发布");
    currentDateText.setText(prefs.getString("current_date", ""));
    weatherInfoLayout.setVisibility(View.VISIBLE);
    cityNameText.setVisibility(View.VISIBLE);
}

}
```


同样，这个活动中的代码也非常长，我们还是一步步梳理下。在 `onCreate()` 方法中仍然是先去获取一些控件的实例，然后会尝试从 `Intent` 中取出县级代号，如果可以取到就会调用 `queryWeatherCode()` 方法，如果不能取到则会调用 `showWeather()` 方法，我们先来看下可以取到的情况。

`queryWeatherCode()` 方法中并没有几行代码，仅仅是拼装了一个地址，然后调用 `queryFromServer()` 方法来查询县级代号所对应的天气代号。服务器返回的数据仍然会回调到 `onFinish()` 方法中，这里对返回的数据进行解析，然后将解析出来的天气代号传入到 `queryWeatherInfo()` 方法中。

`queryWeatherInfo()` 方法也非常简单，同样是拼装了一个地址，然后调用 `queryFromServer()` 方法来查询天气代号所对应的天气信息。由于天气信息是以 JSON 格式返回的，因此我们在 `handleWeatherResponse()` 方法中使用 `JSONObject` 将数据全部解析出来，然后调用 `saveWeatherInfo()` 方法将所有的天气信息都存储到 `SharedPreferences` 文件中。注意除了天气信息之外，我们还存储了一个 `city_selected` 标志位，以此来辨别当前是否已经选中了一个城市。最后会去调用 `showWeather()` 方法来将所有的天气信息显示到界面上，`showWeather()` 方法中的逻辑很简单，就是从 `SharedPreferences` 文件中将数据读取出来，然后一一设置到界面上即可。

刚才分析的是在 `onCreate()` 方法中可以取到县级代号的情况，那么不能取到的时候呢？原来就是直接调用 `showWeather()` 方法来显示本地存储的天气信息就可以了。

那么接下来我们要做的，就是如何从 `ChooseAreaActivity` 跳转到 `WeatherActivity` 了，修改 `ChooseAreaActivity` 中的代码，如下所示：

```
public class ChooseAreaActivity extends Activity {
    .....
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
        if (prefs.getBoolean("city_selected", false)) {
            Intent intent = new Intent(this, WeatherActivity.class);
            startActivity(intent);
            finish();
            return;
        }
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.choose_area);
        listView = (ListView) findViewById(R.id.list_view);
```

```

        titleText = (TextView) findViewById(R.id.title_text);
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_
list_item_1, dataList);
        listView.setAdapter(adapter);
        coolWeatherDB = CoolWeatherDB.getInstance(this);
        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View view, int index,
                long arg3) {
                if (currentLevel == LEVEL_PROVINCE) {
                    selectedProvince = provinceList.get(index);
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    selectedCity = cityList.get(index);
                    queryCounties();
                } else if (currentLevel == LEVEL_COUNTY) {
                    String countyCode = countyList.get(index).getCountyCode();
                    Intent intent = new Intent(ChooseAreaActivity.this,
WeatherActivity.class);
                    intent.putExtra("county_code", countyCode);
                    startActivity(intent);
                    finish();
                }
            }
        });
        queryProvinces(); // 加载省级数据
    }

    .....
}

```

可以看到，这里我们主要修改了两处。第一，在 `onCreate()` 方法的一开始先从 `SharedPreferences` 文件中读取 `city_selected` 标志位，如果为 `true` 就说明当前已经选择过城市了，直接跳转到 `WeatherActivity` 即可。第二，在 `onItemClick()` 方法中加入一个 `if` 判断，如果当前级别是 `LEVEL_COUNTY`，就启动 `WeatherActivity`，并把当前选中县的县级代号传递过去。

最后不要忘了在 `AndroidManifest.xml` 中注册一下新增的活动，如下所示：

```

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

```

```

    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
        <activity android:name="com.coolweather.app.activity.
WeatherActivity"></activity>
    </application>
</manifest>

```

好了，现在运行一下程序，然后选择江苏→苏州→昆山，结果如图 14.19 所示。



图 14.19

OK，这样第三阶段的开发工作也都完成了，我们把代码提交一下。

```

git add .
git commit -m "加入显示天气信息的功能。"
git push origin master

```

14.6 切换城市和手动更新天气

经过第三阶段的开发，现在酷欧天气的主体功能已经有了，不过你会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 WeatherActivity。

因此，在第四阶段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，我们会同时加入手动更新天气的功能。

首先要在布局文件中加入切换城市和更新天气的按钮，修改 weather_layout.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="#484E61" >
        <Button
            android:id="@+id/switch_city"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:layout_centerVertical="true"
            android:layout_marginLeft="10dp"
            android:background="@drawable/home" />
        <TextView
            android:id="@+id/city_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textColor="#fff"
            android:textSize="24sp" />
        <Button
            android:id="@+id/refresh_weather"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true"
```

```

        android:layout_marginRight="10dp"
        android:background="@drawable/refresh" />
    </RelativeLayout>
    .....
</LinearLayout>

```

可以看到，这里添加了两个按钮，一个在标题栏的左边，一个在标题栏的右边，这两个按钮所使用的背景图片我已经提前准备好了。然后修改 WeatherActivity 中的代码，如下所示：

```

public class WeatherActivity extends Activity implements OnClickListener{
    .....

    /**
     * 切换城市按钮
     */
    private Button switchCity;
    /**
     * 更新天气按钮
     */
    private Button refreshWeather;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.weather_layout);
        .....

        switchCity = (Button) findViewById(R.id.switch_city);
        refreshWeather = (Button) findViewById(R.id.refresh_weather);
        switchCity.setOnClickListener(this);
        refreshWeather.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.switch_city:
                Intent intent = new Intent(this, ChooseAreaActivity.class);
                intent.putExtra("from_weather_activity", true);
                startActivity(intent);
                finish();
                break;

```

```

        case R.id.refresh_weather:
            publishText.setText("同步中...");
            SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
            String weatherCode = prefs.getString("weather_code", "");
            if (!TextUtils.isEmpty(weatherCode)) {
                queryWeatherInfo(weatherCode);
            }
            break;
        default:
            break;
    }
}
.....
}

```

我们在 onCreate()方法中获取到了两个按钮的实例，然后分别调用了 setOnClickListener()方法来注册点击事件。当点击的是更新天气按钮时，会首先从 SharedPreferences 文件中读取天气代号，然后调用 queryWeatherInfo()方法去更新天气就可以了。当点击的是切换城市按钮时，会跳转到 ChooseAreaActivity，但是注意目前我们已经选中过了一个城市，如果直接跳转到 ChooseAreaActivity，会立刻又跳转回来，因此这里在 Intent 中加入了一个 from_weather_activity 标志位。

接着在 ChooseAreaActivity 对这个标志位进行处理，如下所示：

```

public class ChooseAreaActivity extends Activity {
    .....
    /**
     * 是否从WeatherActivity中跳转过来。
     */
    private boolean isFromWeatherActivity;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        isFromWeatherActivity = getIntent().getBooleanExtra("from_weather_
activity", false);
        SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
        // 已经选择了城市且不是从WeatherActivity跳转过来，才会直接跳转到
WeatherActivity
    }
}

```

```

        if (prefs.getBoolean("city_selected", false)
&& !isFromWeatherActivity) {
            Intent intent = new Intent(this, WeatherActivity.class);
            startActivity(intent);
            finish();
            return;
        }
        .....
    }
    .....
    @Override
    public void onBackPressed() {
        if (currentLevel == LEVEL_COUNTY) {
            queryCities();
        } else if (currentLevel == LEVEL_CITY) {
            queryProvinces();
        } else {
            if (isFromWeatherActivity) {
                Intent intent = new Intent(this, WeatherActivity.class);
                startActivity(intent);
            }
            finish();
        }
    }
}

```

可以看到，这里我们加入了一个 `isFromWeatherActivity` 变量，以此来标记是不是从 `WeatherActivity` 跳转过来的，只有已经选择了城市且不是从 `WeatherActivity` 跳转过来的时候才会直接跳转到 `WeatherActivity`。另外，我们在 `onBackPressed()` 方法中也进行了处理，当按下 `Back` 键时，如果是从 `WeatherActivity` 跳转过来的，则应该重新回到 `WeatherActivity`。

现在重新运行一下程序，结果如图 14.20 所示。



图 14.20

点击一下标题栏左边的按钮就可以切换城市，点击一下标题栏右边的按钮就可以更新天气，这样我们第四阶段的开发任务也完成了。当然，仍然不要忘记提交代码。

```
git add .  
git commit -m "新增切换城市和手动更新天气的功能。"  
git push origin master
```

14.7 后台自动更新天气

为了要让酷欧天气更加智能，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，这个技术我们在第9章的最佳实践环节就已经学习过了。首先在 service 包下新建一个 AutoUpdateService 继承自 Service，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 startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                updateWeather();
            }
        }).start();
        AlarmManager manager = (AlarmManager) getSystemService(ALARM_SERVICE);
        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, AutoUpdateReceiver.class);
        PendingIntent pi = PendingIntent.getBroadcast(this, 0, i, 0);
        manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtTime, pi);
        return super.onStartCommand(intent, flags, startId);
    }

    /**
     * 更新天气信息。
     */
    private void updateWeather() {
        SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
        String weatherCode = prefs.getString("weather_code", "");
        String address = "http://www.weather.com.cn/data/cityinfo/" +
weatherCode + ".html";
        HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
            @Override
            public void onFinish(String response) {
                Utility.handleWeatherResponse(AutoUpdateService.this,
response);
            }

            @Override
            public void onError(Exception e) {
                e.printStackTrace();
            }
        });
    }

```

```

        });
    }

}

```

可以看到，在 `onStartCommand()` 方法中先是开启了一个子线程，然后在子线程中调用 `updateWeather()` 方法来更新天气，我们仍然会将服务器返回的天气数据交给 `Utility` 的 `handleWeatherResponse()` 方法去处理，这样就可以把最新的天气信息存储到 `SharedPreferences` 文件中。

之后就是我们学习过的创建定时任务的技巧了，为了保证软件不会消耗过多的流量，这里将时间间隔设置为 8 小时，8 小时后就应该执行到 `AutoUpdateReceiver` 的 `onReceive()` 方法中了，在 `receiver` 包下新建 `AutoUpdateReceiver` 继承自 `BroadcastReceiver`，代码如下所示：

```

public class AutoUpdateReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Intent i = new Intent(context, AutoUpdateService.class);
        context.startService(i);
    }

}

```

这里只是在 `onReceive()` 方法中再次去启动 `AutoUpdateService`，就可以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 Activity implements OnClickListener{
    .....

    private void showWeather() {
        SharedPreferences prefs = PreferenceManager.
getDefaultSharedPreferences(this);
        cityNameText.setText( prefs.getString("city_name", ""));
        temp1Text.setText(prefs.getString("temp1", ""));
        temp2Text.setText(prefs.getString("temp2", ""));
        weatherDespText.setText(prefs.getString("weather_desp", ""));
        publishText.setText("今天" + prefs.getString("publish_time", "") +
"发布");
        currentDateText.setText(prefs.getString("current_date", ""));
        weatherInfoLayout.setVisibility(View.VISIBLE);
        cityNameText.setVisibility(View.VISIBLE);
    }
}

```

```

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

```

可以看到，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。

最后，别忘了在 AndroidManifest.xml 中注册新增的服务和广播接收器，如下所示：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.app"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
        <service android:name="com.coolweather.app.service.
AutoUpdateService"></service>
        <receiver android:name="com.coolweather.app.receiver.
AutoUpdateReceiver"></receiver>
    </application>
</manifest>

```

现在可以再提交一下代码：

```

git add .
git commit -m "增加后台自动更新天气的功能。"
git push origin master

```

14.8 修改图标和名称

目前的酷欧天气看起来还不太像是一个正式的软件，为什么呢？因为都还没有一个像样的图标呢。一直使用 ADT 自动生成的图标确实不太合适，是时候需要换一下了。

这里我事先准备好了一张图片来作为软件图标，由于我也不是搞美术的，因此图标设计得非常简单，如图 14.21 所示。



图 14.21

将这张图片命名成 logo.png，放入 res/drawable-hdpi 目录，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.app"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
    </application>
</manifest>
```

这里将<application>标签的 android:icon 属性指定成 logo.png 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，将它修改成酷欧天气即可，如下所示：

```
<resources>
    <string name="app_name">酷欧天气</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

由于更改了程序图标和名称，我们需要先将酷欧天气卸载掉，然后重新运行一遍程序，这时观察酷欧天气的桌面图标，如图 14.22 所示。



图 14.22

养成良好的习惯，仍然不要忘记提交代码。

```
git add .
git commit -m "修改程序图标和名称。"
git push origin master
```

这样我们就终于大功告成了！

14.9 你还可以做的事情

经过五个阶段的开发，酷欧天气已经是一个完善、成熟的软件了吗？嘿嘿，还差得远呢！现在的酷欧天气只能说是具备了一些最基本的功能，和那些商用的天气软件比起来还有很大的差距，因此你仍然还有非常巨大的发挥空间来对它进行完善。

比如说以下功能是你可以考虑加入到酷欧天气中的。

1. 增加设置选项，让用户选择是否允许后台自动更新天气，以及设定更新的频率。
2. 优化软件界面，提供多套与天气对应的图片，让程序可以根据不同的天气自动切换背景图。
3. 允许选择多个城市，可以同时观察多个城市的天气信息，不用来回切换。
4. 提供更加完整的天气信息，包括未来几天的天气情况、风力指数、生活建议等。

其中，第四项功能使用目前的查询天气接口是无法完成的，因为这个接口得不到如此详细的数据。不过没有关系，我们可以改用另外一个接口，如下所示：

<http://m.weather.com.cn/data/101190404.html>

101190404 是昆山所对应的天气代号，如果想要查看其他城市的天气信息只需要改成相应城市的天气代码就可以了。这个接口返回的数据非常详细，你在浏览器中访问一下就知道了。

另外，由于酷欧天气的源码已经托管在了 GitHub 上面，如果你想在现有代码的基础上继续对这个项目进行完善，就可以使用 GitHub 的 Fork 功能。

首先登录你自己的 GitHub 账号，然后打开酷欧天气版本库的主页：

<https://github.com/tony-green/coolweather>

这时在页面头部的最右侧会有一个 Fork 按钮，如图 14.23 所示。



图 14.23

点击一下 Fork 按钮就可以将酷欧天气这个项目复制一份到你的账号下，再使用 `git clone` 命令将它克隆到本地，然后你就可以在现有代码的基础上随心所欲地添加任何功能并提交了。

经验值：+1000000 升级！（由巨鹰升级至神鹰） 目前经验值：1864905

级别：神鹰

赢得宝物：战胜烛龙。烛龙，人面蛇身，通体赤红，身长千里，他发出的强大光芒能照亮最黑暗的地方，是神界最尊贵的三位开天辟地级大神之一，也是神界五位不可战胜神之一。烛龙是不可战胜的，所谓战胜只是顶住了它二分功力持续一秒的光压。作为奖励，烛龙对我的翅膀进行了转化，将肉身的翅膀转变为光芒之翅，可随意念收展自如。当把光芒之翅收起时，我与常人无异，当我展开光芒之翅时，翼展十米的光的翅膀在我的背部打开，飞行速度已提高三倍。拜别烛龙，我拍击新的光芒之翅，腾空而起，如一道光芒，向前方飞去。