2021

第12章 综合示例设计与开发

复旦大学 陈辰

学习目标

AIMS

01 掌握Android应用程序的基本设计方法和思路

02 掌握使用多种组件进行Android程序开发的方法

12.1 需求分析

- **设计本章的初衷**
  - 通过前面章节的学习，读者应该已经掌握了一些Android应用程序的开发知识和方法，但如何综合的运用这些知识和方法，解决实际开发中所遇到的问题，还是一个需要继续学习和探讨的问题。设计本章的初衷就是希望读者能够根据实际项目的需求，准确的分析出Android应用程序开发所可能涉及到的知识点，通过分析软件的需求，快速设计出用户界面和模块结构，并最终完成应用程序的开发和调试。

12.1 需求分析

- **功能需求**
  - 本章提供的“天气预报软件”是一个略微复杂的示例。在这个综合示例中，有一个显示天气情况的用户界面，可以通过图片和文字显示当前和未来几天的天气状况，包括温度、湿度、风向和雨雪情况等。这些天气数据是通过后台服务获取的，这个后台服务按照一定时间间隔，从中国天气网上获取天气预报信息，并将天气信息保存在后台服务中。示例还需要提供基于SMS短信的天气数据服务，其他手机用户可以向本示例所在的手机上发送SMS短信，在短信中包含特定的关键字，则可以将已有的天气情况通过SMS短信回复给用户。最后，每个被发送的SMS短信都会被记录下来，用户可以浏览或删除这些记录信息。

12.1 需求分析

- **界面需求**
  - 从上面的描述中可以基本了解软件的功能需求，但为了将需求分析过程变得简单明了，首先找出用户界面上需要显示的内容。功能描述中有“显示天气情况的用户界面”和“用户可以浏览或删除这些记录信息”，除此以外，一般应用软件还应有显示配置信息的界面。因此，本示例应该包含三个用户界面：
    - 显示天气预报的用户界面
    - 显示已发送SMS短信的用户界面
    - 浏览和设置配置信息的用户界面

12.1 需求分析

- **内部功能**
  - 从用户界面出发，分析隐藏在界面后面的内部功能，这些功能则是程序正常运行的基础
    - 显示天气预报的用户界面
      - 获取中国天气网的天气数据
      - 保存天气数据信息
    - 显示SMS短信的用户界面
      - 根据关键字监视SMS短信
      - 发送包含天气信息的SMS短信
      - 将发送SMS短信的相关信息写入数据库
    - 浏览和设置配置信息的用户界面
      - 将用户设置的配置信息保存到数据库
      - 启动时读取数据库中的配置信息
      - 支持恢复缺省设置

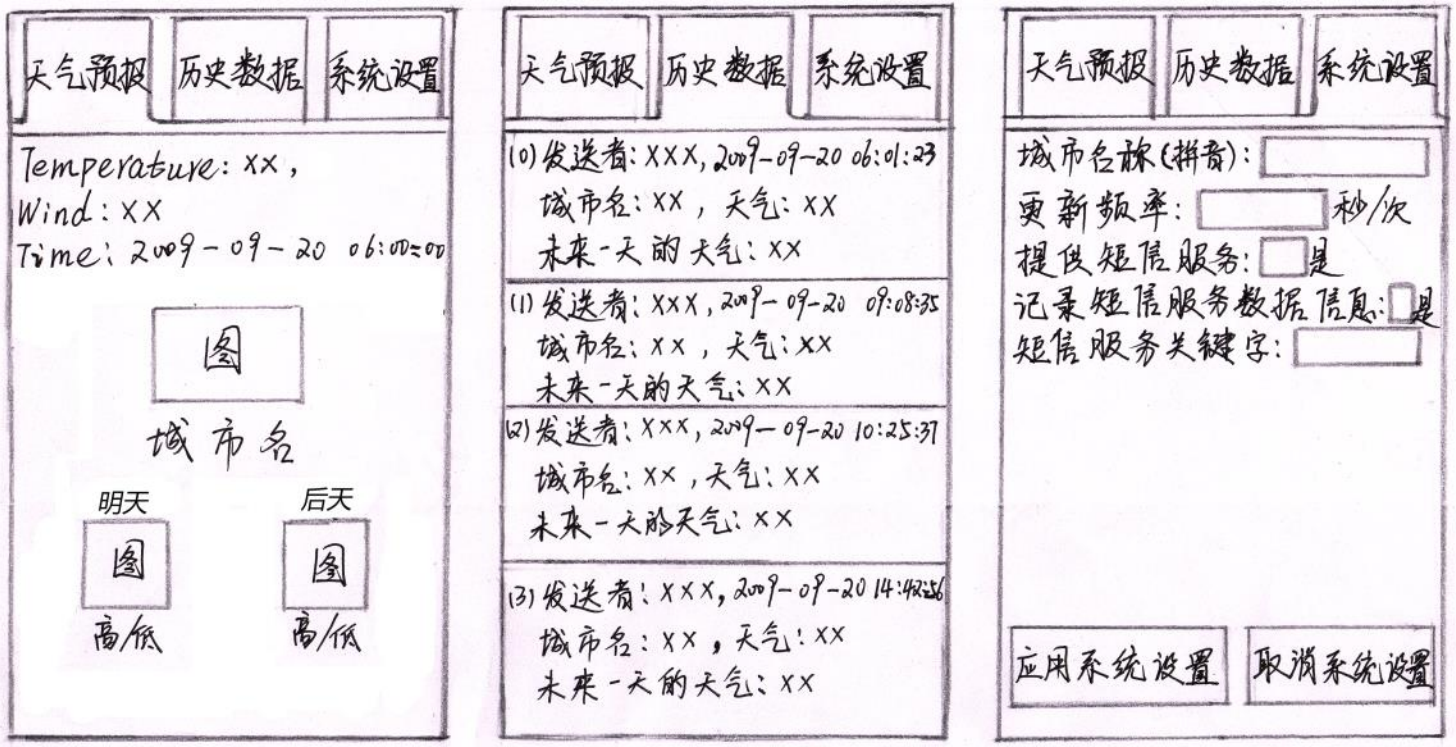
12.2 程序设计

- **12.2.1 用户界面设计**
  - 根据需求中的用户界面分析，应用程序应包含三个主要的用户界面，每个用户界面的显示内容
    - 在“显示天气预报的用户界面”中，显示目标城市的当前的天气状况，包括城市名称、温度、湿度、风向、雨雪情况和获取数据时间等信息。在界面的下方显示未来四天的天气状况，但仅包括温度和雨雪情况
    - 在“显示已发送SMS短信的用户界面”中，显示每个回复短信的时间、目标手机号码、城市名称、当天的天气状况和未来一天的天气状况
    - 在“浏览和设置配置信息的用户界面”中，显示希望获取天气预报的城市名称、获取数据的频率和短信监视关键字，并允许用户设置是否提供短信服务，以及是否记录回复短信信息

12.2 程序设计

• 12.2.1 用户界面设计

- 用户界面的草图



12.2 程序设计

- **12.2.2 数据库设计**
  - 本示例主要有两种数据需要存储
    - 配置信息：因为配置信息的数据量很小，从Android支持的存储方式上分析，可以保存在SharedPreference、文件或SQLite数据库中
    - SMS短信服务信息： SMS短信服务信息是一个随着时间推移而不断增加的数据，属于文本信息，而且有固定的格式，因此适合使用SQLite数据库进行存储
  - 综合分析这两个需要存储的数据，选择SQLite数据库作为存储数据的方法

12.2 程序设计

• 12.2.2 数据库设计

- 配置信息
  - 配置信息中主要保存目标城市的名称，访问中国天气网更新天气信息的频率，请求天气信息服务短信的关键字，以及是否提供短信服务和是否记录短信服务内容
  - 配置信息的数据库表结构

属性	数据类型	说明
_id	integer	自动增加的主键
city_name	text	天气信息查询的城市名
refresh_speed	text	天气信息查询的频率，单位为秒/次
sms_service	text	是否提供短信服务，即接收到请求短信后是否回复包含天气信息的短信
sms_info	text	是否记录发出的SMS短信的信息
key_word	text	短信服务的关键字，用以确定哪条短信是请求天气服务的短信

12.2 程序设计

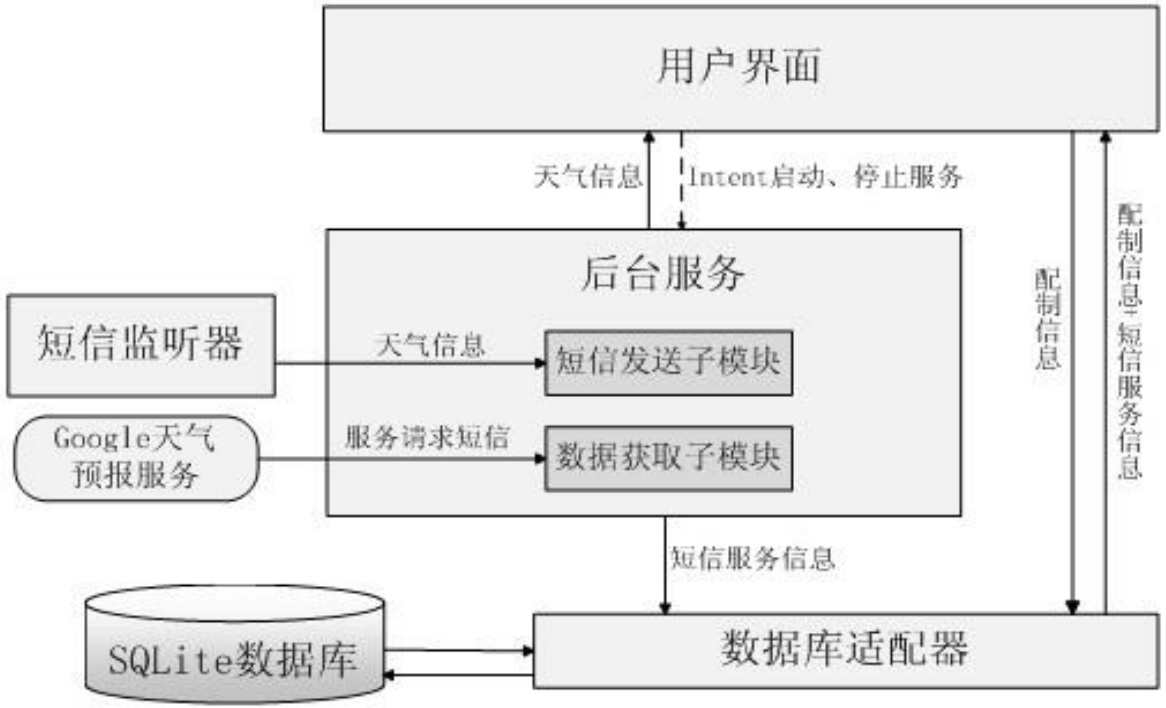
• 12.2.2 数据库设计

- SMS短信服务信息
  - SMS短信服务信息主要保存请求天气服务短信的发送者、短信内容、接收时间和回复信息的内容
  - SMS短信服务信息的数据库表结构

属性	数据类型	说明
_id	integer	自动增加的主键
sms_sender	text	请求服务短信的发送者
sms_body	text	请求服务短信的内容信息
sms_receive_time	text	接收到请求服务短信的时间
return_result	text	回复短信的内容

12.2 程序设计

- **12.2.3 程序模块设计**
  - 从功能需求上分析，可以将整个应用程序划分为4个模块，分别是用户界面、后台服务、数据库适配器和短信监听器 。下图是模块结构图:



12.2 程序设计

- **12.2.3 程序模块设计**
  - 从模块结构图中不难看出，后台服务是整个应用程序的核心，主要包含数据获取子模块和短信服务子模块。数据获取子模块负责周期性从中国天气网获取天气信息；短信服务子模块则负责处理接收到的服务请求短信，并发送包含天气信息的短信
  - 后台服务由用户界面通过Intent启动，启动后的后台服务可以在用户界面关闭后仍然保持运行状态，直到用户通过用户界面发送Intent停止服务，或系统因资源不足而强行关闭服务

12.2 程序设计

- **12.2.3 程序模块设计**
  - 用户界面从后台服务获取天气信息，而没有直接通过网络访问中国天气网的天气数据
    - 一方面是因为后台服务使用了工作线程，通过后台服务获取天气数据可以避免因网络通信不畅造成界面失去响应
    - 另一方面，在用户关闭界面后，后台服务仍然需要更新天气信息，以保证短信服务数据的准确性。用户界面通过直接调用数据库适配器，向SQLite数据库中读写配置信息，或对SMS短信服务信息进行操作

12.2 程序设计

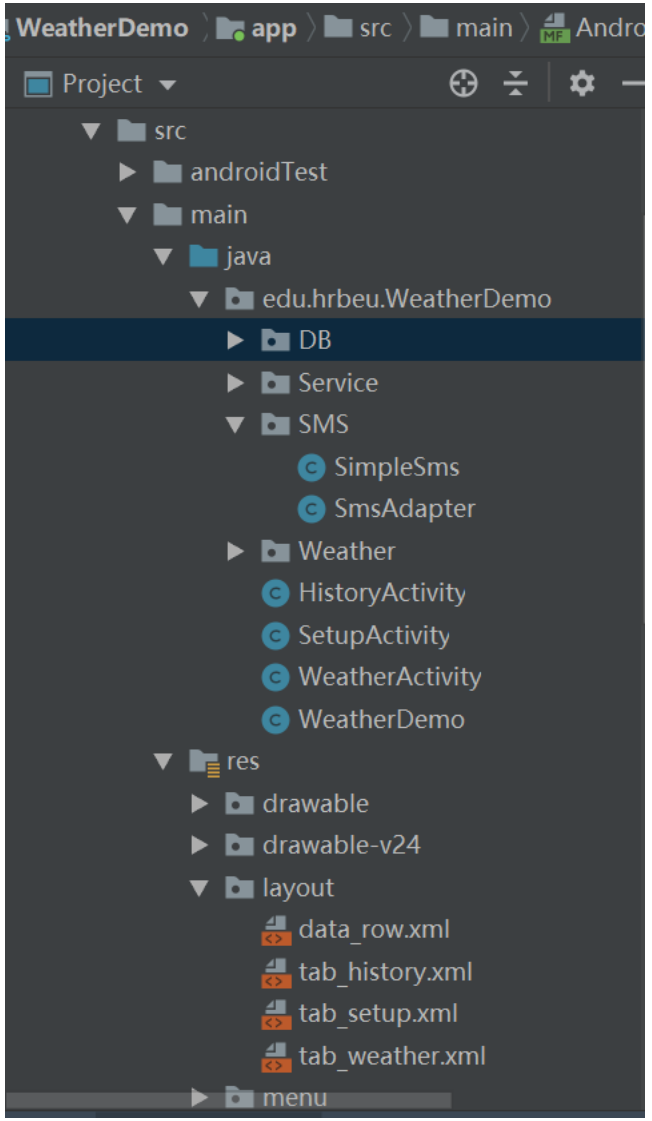
- **12.2.3 程序模块设计**
  - 短信监听器是一个BroadcastReceiver，监视所有接收到的短信
    - 如果短信中包含用户自定义的关键字，短信监听器则会认为这条短信是天气服务请求短信，将短信的相关信息写入后台服务的短信服务队列
    - 如果用户在配置信息中选择无需提供短信服务，短信监听器仍然继续监听所有短信，只是后台服务不再允许将服务请求短信写入服务队列
  - 数据库适配器封装了所有对SQLite数据库操作的方法，用户界面和后台服务会调用它实现数据库操作

12.3 程序开发

- **12.3.1 工程结构**
  - 在程序开发阶段，首先确定“天气预报软件”的工程名称为WeatherDemo，包名称为edu.hrbeu.WeatherDemo。然后根据程序模块设计的内容，建立WeatherDemo示例

12.3 程序开发

- **12.3.1 工程结构**
  - WeatherDemo示例源代码的文件结构



12.3 程序开发

• 12.3.1 工程结构

- 为了使源代码文件的结构更加清晰，WeatherDemo示例设置了多个命名空间中，分别用来保存用户界面、数据库、后台服务、SMS短信和天气数据的源代码文件
- WeatherDemo示例的命名空间

命名空间	说明
edu.hrbeu.WeatherDemo	用户界面相关的源代码文件
edu.hrbeu.WeatherDemo.DB	SQLite数据库相关的源代码文件
edu.hrbeu.WeatherDemo.Service	后台服务相关的源代码文件
edu.hrbeu.WeatherDemo.SMS	SMS短信相关的源代码文件
edu.hrbeu.WeatherDemo.Weather	天气数据有关的源代码文件

12.3 程序开发

• 12.3.1 工程结构

- WeatherDemo示例将不同用途的源代码文件放置在不同的命名空间中
- WeatherDemo示例的文件用途说明

包名称	文件名	说明
.WeatherDemo	HistoryActivity.java	“历史数据”页的Activity
	SetupActivity.java	“系统设置”页的Activity
	WeatherActivity.java	“天气预报”页的Activity
	WeatherDemo.java	程序启动缺省的Activity
.WeatherDemo.DB	Config.java	配置信息的类
	DBAdapter.java	数据库适配器
.WeatherDemo.Service	SmsReceiver.java	短信监听器
	WeatherAdapter.java	数据获取模块
	WeatherService.java	后台服务
.WeatherDemo.SMS	SimpleSms.java	简化的SMS短信类
	SmsAdapter.java	短信发送模块
.WeatherDemo.Weather	Forecast.java	天气信息的类

12.3 程序开发

- **12.3.1 工程结构**
  - Android资源文件保存在/res的子目录中
    - /res/drawable目录中保存的是图像文件
    - /res/layout目录中保存的是布局文件
    - /res/values目录中保存的是用来定义字符串和颜色的文件
  - 所有在程序开发阶段可以被调用的资源都保存在这些目录中

12.3 程序开发

• 12.3.1工程结构

- 资源文件名称与用途

资源目录	文件	说明
drawable	icon.png	图标文件
	sunny.png	调试用的天气图片
	tab_history.png	“历史数据”页的图片
	tab_setup.png	“系统设置”页的图片
	tab_weather.png	“天气预报”页的图片
layout	data_row.xml	“历史数据”页ListActivity的每行数据的布局
	tab_history.xml	“历史数据”页的布局
	tab_setup.xml	“系统设置”页的布局
	tab_weather.xml	“天气预报”页的布局
values	color.xml	保存颜色的XML文件
	string.xml	保存字符串的XML文件
xml	api.xml	从Google下载的天气数据文件。在程序运行时没有实际作用，但在开发过程中可以让读者了解数据格式

12.3 程序开发

- **12.3.2 数据库适配器**
  - 数据库适配器是最底层的模块，主要用于封装用户界面和后台服务对SQLite数据库的操作。数据库适配器的核心代码主要在DBAdapter.java文件中
  - 用户保存配置信息的类文件Config.java
  - Config.java文件的全部代码如下：

```
1   package edu.hrbeu.WeatherDemo.DB;
2
3   public class Config {
4       public static String CityName;
5       public static String RefreshSpeed;
6       public static String ProvideSmsService;
7       public static String SaveSmsInfo;
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - Config.java文件代码

```
8       public static String KeyWord;
9
10        public static void LoadDefaultConfig(){
11            CityName = "101050101";
12            RefreshSpeed = "60";
13            ProvideSmsService = "true";
14            SaveSmsInfo = "true";
15            KeyWord = "NY";
16        }
17    }
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - 从代码中不难看出，公有静态属性CityName、RefreshSpeed、ProvideSmsService、SaveSmsInfo和KeyWord完全对应数据库中保存配置信息表的属性(参照表12.1)。在程序启动后，保存数据库中的城市名称、更新频率、是否提供短信服务、是否保存短信信息和关键字等内容，将被读取到这个Config类中，供其它模块做逻辑判断时使用
    - 代码第10行的LoadDefaultConfig()函数，保存了程序内置的配置参数

12.3 程序开发

- **12.3.2 数据库适配器**
  - 此函数会在两个情况下被调用
    - 用户主动选择“恢复缺省设置”
    - 首次启动程序时，用来初始化保存配置参数的数据库
  - DBAdapter类与以往介绍过的数据库适配器类相似，都具有继承SQLiteOpenHelper的帮助类DBOpenHelper
  - DBOpenHelper在建立数据库时，同时建立两个数据库表，并对保存配置信息的表进行了初始化，初始化的相关代码在第42 ~ 49行

12.3 程序开发

• 12.3.2 数据库适配器

- 数据库表代码

```
1   private static final String DB_NAME = "weather_app.db";
2   private static final String DB_TABLE_CONFIG = "setup_config";
3   private static final String DB_CONFIG_ID = "1";
4   private static final int DB_VERSION = 1;
5
6   public static final String KEY_ID = "_id";
7   public static final String KEY_CITY_NAME = "city_name";
8   public static final String KEY_REFRESH_SPEED = "refresh_speed";
9   public static final String KEY_SMS_SERVICE = "sms_service";
10   public static final String KEY_SMS_INFO = "sms_info";
11   public static final String KEY_KEY_WORD = "key_word";
12
13   private static final String DB_TABLE_SMS = "sms_data";
14   public static final String KEY_SENDER = "sms_sender";
15   public static final String KEY_BODY = "sms_body";
```

12.3 程序开发

• 12.3.2 数据库适配器

- 数据库表代码

```
16  public static final String KEY_RECEIVE_TIME = "sms_receive_time";
17  public static final String KEY_RETURN_RESULT = "return_result";
18
19  /**静态Helper类，用于建立、更新和打开数据库*/
20  private static class DBOpenHelper extends SQLiteOpenHelper {
21  public DBOpenHelper(Context context, String name, CursorFactory factory, int
      version) {
22        super(context, name, factory, version);
23        }
24
25     private static final String DB_CREATE_CONFIG = "create table " +
26    DB_TABLE_CONFIG + " (" + KEY_ID + " integer primary key autoincrement, " +
27        KEY_CITY_NAME+ " text not null, " + KEY_REFRESH_SPEED+ " text," +
28        KEY_SMS_SERVICE +" text, " + KEY_SMS_INFO +  " text, " +
29        KEY_KEY_WORD + " text);";
```

12.3 程序开发

• 12.3.2 数据库适配器

- 数据库表代码

```
30
31      private static final String DB_CREATE_SMS = "create table " +
32          DB_TABLE_SMS + " (" + KEY_ID + " integer primary key autoincrement, " +
33          KEY_SENDER+ " text not null, " + KEY_BODY+ " text, " +
34          KEY_RECEIVE_TIME +" text, " +   KEY_RETURN_RESULT + " text);";
35
36      @Override
37      public void onCreate(SQLiteDatabase _db) {
38          _db.execSQL(DB_CREATE_CONFIG);
39          _db.execSQL(DB_CREATE_SMS);
40
41          //初始化系统配置的数据表
42          Config.LoadDefaultConfig();
43          ContentValues newValues = new ContentValues();
44          newValues.put(KEY_CITY_NAME, Config.CityName);
45          newValues.put(KEY_REFRESH_SPEED, Config.RefreshSpeed);
```

12.3 程序开发

• 12.3.2 数据库适配器

- 数据库表代码

```
46          newValues.put(KEY_SMS_SERVICE, Config.ProvideSmsService);
47          newValues.put(KEY_SMS_INFO, Config.SaveSmsInfo);
48          newValues.put(KEY_KEY_WORD, Config.KeyWord);
49          _db.insert(DB_TABLE_CONFIG, null, newValues);
50      }
51
52      @Override
53      public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
54          _db.execSQL("DROP TABLE IF EXISTS " + DB_TABLE_CONFIG);
55          _db.execSQL("DROP TABLE IF EXISTS " + DB_CREATE_SMS);
56          onCreate(_db);
57      }
58  }
```

12.3 程序开发

• 12.3.2 数据库适配器

- 在DBAdapter类中，用户界面会调用SaveConfig()和LoadConfig()，从SQLite数据库中保存和读取配置信息。保存配置信息时，SaveConfig()函数会将Config类中的公有静态属性写入数据库；反之，LoadConfig()会将数据库中的配置信息写入Config类中的公有静态属性
- SaveConfig()和LoadConfig()函数的代码如下：

```
1   public void SaveConfig(){
2       ContentValues updateValues = new ContentValues();
3       updateValues.put(KEY_CITY_NAME, Config.CityName);
4       updateValues.put(KEY_REFRESH_SPEED, Config.RefreshSpeed);
5       updateValues.put(KEY_SMS_SERVICE, Config.ProvideSmsService);
6       updateValues.put(KEY_SMS_INFO, Config.SaveSmsInfo);
7       updateValues.put(KEY_KEY_WORD, Config.KeyWord);
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - SaveConfig()和LoadConfig()函数代码

```
8       db.update(DB_TABLE_CONFIG, updateValues,  KEY_ID + "=" +
        DB_CONFIG_ID, null);
9       Toast.makeText(context, "系统设置保存成功", Toast.LENGTH_SHORT).show();
10    }
11
12    public void LoadConfig() {
13       Cursor result =  db.query(DB_TABLE_CONFIG, new String[] { KEY_ID,
        KEY_CITY_NAME,
14          KEY_REFRESH_SPEED,KEY_SMS_SERVICE, KEY_SMS_INFO,
        KEY_KEY_WORD},
15          KEY_ID + "=" + DB_CONFIG_ID, null, null, null, null);
16       if (result.getCount() == 0 || !result.moveToFirst()){
17             return;
18          }
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - SaveConfig()和LoadConfig()函数代码

```
19      Config.CityName = result.getString(result.getColumnIndex(KEY_CITY_NAME));
20      Config.RefreshSpeed =
    result.getString(result.getColumnIndex(KEY_REFRESH_SPEED));
21
    Config.ProvideSmsService=result.getString(result.getColumnIndex(KEY_SMS_SE
    RVICE));
22      Config.SaveSmsInfo =
    result.getString(result.getColumnIndex(KEY_SMS_INFO));
23      Config.KeyWord = result.getString(result.getColumnIndex(KEY_KEY_WORD));
24      Toast.makeText(context, "系统设置读取成功", Toast.LENGTH_SHORT).show();
25  }
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - 另一个会调用DBAdapter类的是后台服务，即WeatherService类
  - 后台服务主要调用SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数，分别用来保存SMS短信记录、删除所有SMS数据记录和获取所有SMS数据记录
  - 在GetAllSms()函数中，调用了一个私有函数ToSimpleSms(Cursor cursor)，用来将从数据库获取的数据转换为SimpleSms实例数组
  - SimpleSms类将在下一小节进行介绍

12.3 程序开发

- **12.3.2 数据库适配器**
  - 下面先给出SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数的代码

```
1   public void SaveOneSms(SimpleSms sms){
2       ContentValues newValues = new ContentValues();
3       newValues.put(KEY_SENDER, sms.Sender);
4       newValues.put(KEY_BODY, sms.Body);
5       newValues.put(KEY_RECEIVE_TIME, sms.ReceiveTime);
6       newValues.put(KEY_RETURN_RESULT, sms.ReturnResult);
7       db.insert(DB_TABLE_SMS, null, newValues);
8   }
9   public long DeleteAllSms() {
10        return db.delete(DB_TABLE_SMS, null, null);
11    }

12    public SimpleSms[] GetAllSms() {
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数代码

```
13      Cursor results =  db.query(DB_TABLE_SMS, new String[] { KEY_ID,
        KEY_SENDER,
14          KEY_BODY, KEY_RECEIVE_TIME, KEY_RETURN_RESULT},
15          null, null, null, null, null);
16       return ToSimpleSms(results);
17   }
18   private SimpleSms[] ToSimpleSms(Cursor cursor){
19      int resultCounts = cursor.getCount();
20      if (resultCounts == 0 || !cursor.moveToFirst()){
21          return null;
22      }
23
24      SimpleSms[] sms = new SimpleSms[resultCounts];
```

12.3 程序开发

- **12.3.2 数据库适配器**
  - SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数代码

```
25      for (int i = 0 ; i<resultCounts; i++){
26          sms[i] = new SimpleSms();
27          sms[i].Sender = cursor.getString(cursor.getColumnIndex(KEY_SENDER));
28          sms[i].Body = cursor.getString(cursor.getColumnIndex(KEY_BODY));
29          sms[i].ReceiveTime =
        cursor.getString(cursor.getColumnIndex(KEY_RECEIVE_TIME));
30
        sms[i].ReturnResult=cursor.getString(cursor.getColumnIndex(KEY_RETURN_RES
        ULT));
31          cursor.moveToNext();
32      }
33      return sms;
34  }
```

12.3 程序开发

- **12.3.3 短信监听器**
  - 短信监听器本质上是BroadcastReceiver，用于监听Android系统所接收到的所有SMS短消息，可以在应用程序关闭后仍然继续运行，核心代码在SmsReceiver.java文件中
  - 在介绍SmsReceiver类前，先说明用来保存SMS短信内容和相关信息的SimpleSms类。android.telephony.SmsMessage是Android提供的短信类，但这里需要一个更精简、小巧的类，保存少量的信息，因此构造了SimpleSms类，仅用来保存短信的发送者、内容、接收时间和返回结果。这里的“返回结果”指的是返回包含天气信息的短信内容

12.3 程序开发

• 12.3.3 短信监听器

- SimpleSms.java文件完整代码

```
1   package edu.hrbeu.WeatherDemo.SMS;
2   import java.text.SimpleDateFormat;
3
4   public class SimpleSms {
5       public String Sender;
6       public String Body;
7       public String ReceiveTime;
8       public String ReturnResult;
9
10       public SimpleSms(){
11       }
12       public SimpleSms(String sender, String body){
13           this.Sender = sender;
14           this.Body = body;
```

12.3 程序开发

• 12.3.3 短信监听器

- SimpleSms.java文件完整代码

```
15          SimpleDateFormat tempDate = new SimpleDateFormat("yyyy-MM-dd" + " " +
        "hh:mm:ss");
16          this.ReceiveTime = tempDate.format(new java.util.Date());
17          this.ReturnResult = "";
18      }
19  }
```

- 代码第5行到第8行的属性Sender、Body、ReceiveTime和ReturnResult，分别表示SMS短信的发送者、内容、接收时间和返回结果
- 第15行和第16行在SimpleSms类的构造函数中，直接将系统时间以“年-月-日 小时:分:秒”的格式保存在ReceiveTime属性中

12.3 程序开发

- **12.3.3 短信监听器**
  - SmsReceiver类继承BroadcastReceiver，重载了onReceive()函数
  - 系统消息的识别和关键字的识别并不复杂，只要接收android.provider.Telephony.SMS_RECEIVED类型的系统消息，则表明是Android系统接收到了短信。将短信的内容拆分后，判断消息内容是否包含了用户定义的关键字，则可判断该短信是否为天气服务请求短信

12.3 程序开发

• 12.3.3 短信监听器

- SmsReceiver.java文件的核心代码

```
1   public class SmsReceiver extends BroadcastReceiver{
2       private static final String SMS_ACTION =
        "android.provider.Telephony.SMS_RECEIVED";
3
4       @Override
5       public void onReceive(Context context, Intent intent){
6           if (intent.getAction().equals(SMS_ACTION)){
7               Bundle bundle = intent.getExtras();
8               if (bundle != null){
9                   Object[] objs = (Object[]) bundle.get("pdus");
10                  SmsMessage[] messages = new SmsMessage[objs.length];
11                  for (int i = 0; i<objs.length; i++){
12                      messages[i] = SmsMessage.createFromPdu((byte[]) objs[i]);
13                  }
```

12.3 程序开发

- **12.3.3 短信监听器**
  - SmsReceiver.java文件代码

```
14              String smsBody = messages[0].getDisplayMessageBody();
15              String smsSender =  messages[0].getDisplayOriginatingAddress();
16              if (smsBody.trim().equals(Config.KeyWord) &&
    Config.ProvideSmsService.equals("true")){
17                  SimpleSms simpleSms = new SimpleSms(smsSender, smsBody);
18                  WeatherService.RequerSMSService(simpleSms);
19                  Toast.makeText(context, "接收到服务请求短信",
    Toast.LENGTH_SHORT).show();
20              }
21          }
22      }
23    }
24  }
```

12.3 程序开发

- **12.3.3 短信监听器**
  - 代码第9行将带有pdus字符串特征的对象，通过bundle.get()函数提取出来
  - 在代码第12行使用SmsMessage.CreateFromPdu()函数构造SmsMessage实例
  - 在代码第12行使用循环语句是因为接收到的短信可能不止一条
  - 从第14行和第15行代码上看，这里只处理第1条短信
  - 第17行构造SimpleSms实例
  - 在代码第18行调用WeatherService类的RequerSMSService()函数，将SimpleSms实例添加到短信队列中

12.3 程序开发

- **12.3.3 短信监听器**
  - 最后，在AndroidManifest.xml文件中注册短信监听器SmsReceiver，并声明可接收短信的用户许可android.permission.RECEIVE_SMS
  - 如果注册的组件不在根命名空间中，则需要将子命名空间写在类的前面

12.3 程序开发

- **12.3.3 短信监听器**
  - 例如下面在代码第1行中，因为SmsReceiver.java文件在edu.hrbeu.WeatherDemo.Service命名空间下，而不在根命名空间edu.hrbeu.WeatherDemo下，因此注册组件时需要在类名SmsReceiver前添加.Service

```
1  <receiver android:name=".Service.SmsReceiver" >
2      <intent-filter>
3          <action android:name="android.provider.Telephony.SMS_RECEIVED" />
4      </intent-filter>
5  </receiver>
6  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

12.3 程序开发

- **12.3.4 后台服务**
  - 后台服务是WeatherDemo示例的核心模块，在用户启动后持续在后台运行，直到用户手动停止服务
  - 后台服务功能
    - 发送包含天气信息的SMS短信（短信发送子模块）
    - 周期性的获取中国天气网的天气数据（数据获取子模块）

12.3 程序开发

- **12.3.4 后台服务**
  - 短信发送子模块： 后台服务在单独的线程上运行
    - 首先调用ProcessSmsList()函数，检查短信队列中是否有需要回复的短信
    - 然后调用GetWeatherData()函数获取天气数据
    - 最后线程暂停1秒，以释放CPU资源
  - WeatherDemo示例后台服务的核心代码在WeatherService.java文件中

12.3 程序开发

• 12.3.4 后台服务

- 下面是线程调用函数的部分代码

```
1   private static ArrayList<SimpleSms> smsList = new ArrayList<SimpleSms>();
2
3   private Runnable backgroudWork = new Runnable(){
4       @Override
5       public void run() {
6           try{
7               while(!Thread.interrupted()){
8                   ProcessSmsList();
9                   GetWeatherData();
10                   Thread.sleep(1000);
11              }
12          } catch (InterruptedException e) {
13              e.printStackTrace();
14          }
15      }
16  };
```

12.3 程序开发

• 12.3.4 后台服务

- ProcessSmsList()函数用来检查短信列表smsList，并根据Weather类中保存的天气数据，向请求者的发送回复短信
- WeatherService.java文件的ProcessSmsList()函数代码如下：

```
1   private void ProcessSmsList(){
2       if (smsList.size()==0){
3           return;
4       }
5       SmsManager smsManager = SmsManager.getDefault();
6       PendingIntent mPi = PendingIntent.getBroadcast(this, 0, new Intent(), 0);
7       while(smsList.size()>0){
8           SimpleSms sms = smsList.get(0);
```

12.3 程序开发

- **12.3.4 后台服务**
  - WeatherService.java文件的ProcessSmsList()函数代码

```
9            smsList.remove(0);
10           smsManager.sendTextMessage(sms.Sender, null, Weather.GetSmsMsg(),
        mPi, null);
11           sms.ReturnResult = Weather.GetSmsMsg();
12           SaveSmsData(sms);
13       }
14   }
```

12.3 程序开发

- **12.3.4 后台服务**
  - 发送短信是使用SmsManager对象的sendTextMessage()方法，该方法一共需要5个参数
    - 第1个参数是收件人地址
    - 第2个参数是发件人地址
    - 第3个参数是短信正文
    - 第4个参数是发送服务
    - 第5个参数是送达服务
  - sendTextMessage()方法的收件人地址和短信正文是不可为空的参数，而且一般GSM规范要求短信内容要控制在70个汉字以内
  - 代码第8行的Weather.GetSmsMsg()，用来获得供回复短信使用的天气信息，因为考虑到短信的字数限制，仅返回当天和未来一天的天气状况

12.3 程序开发

- **12.3.4 后台服务**
  - Weather.java文件的代码如下：

```
1   package edu.hrbeu.WeatherDemo.Weather;
2
3   public class Weather {
4
5       public static String city;// c3
6       public static String province;// c7
7       public static String date;// f0
8
9       public static String current_weather;
10
11      public static String current_temperature;
12      public static String current_windD;
13
14      public static String current_windP;
15
```

12.3 程序开发

- **12.3.4 后台服务**
  - Weather.java文件代码

```
16    public static String[] weatherD = new String[3];// fa
17    public static String[] weatherN = new String[3];// fb18
18
19    public static String[] temperatureD = new String[3];// fc
20    public static String[] temperatureN = new String[3];// fd
21    public static String[] humidity = new String[3];
22    public static String[] windDD = new String[3];// fe
23    public static String[] windDN = new String[3];// ff
24
25    public static String[] windPD = new String[3];// fg
26    public static String[] windPN = new String[3];// fh
27
28     public static String[] sunrise = new String[3];// fi
29     public static String[] sundown = new String[3];//
30
31     public static String GetSmsMsgD(){
```

12.3 程序开发

- **12.3.4 后台服务**
  - Weather.java文件代码

```
32          String msg = "";
33          msg += city + "，  ";
34          msg += weatherD[0] + ", " + temperatureD[0]+". ";
35  //      msg += day[0].day_of_week+", " + day[0].condition + ", " +
36  //                                          day[0].high + "/" + day[0].low;
37          return msg;
38      }
39      public static String GetSmsMsgN(){
40          String msg = "";
41          msg += city + "，  ";
42          msg += weatherN[0] + ", " + temperatureN[0]+". ";
43  //      msg += day[0].day_of_week+", " + day[0].condition + ", " +
44  //                                          day[0].high + "/" + day[0].low;
45          return msg;
46      }
47
48  }
```

12.3 程序开发

- **12.3.4 后台服务**
  - 天气数据是从中国天气网提供的气象数据开放平台中获取的，使用气象数据开放平台API,首先需要申请个人key。调试WeatherDemo示例时需要网络环境，数据的获取地址是： http://openweather.weather.com.cn/
  - 通过查阅提供的API文档，可知其天气调用方式为使用GET请求，获取地址格式为：
    http://open.weather.com.cn/data/?areaid=""&type=""&date=""&appid=""&key=""
    - areaid表示获取天气数据城市的区域id,appid与key再申请后可获得。最终返回的数据格式为JSON格式，可免费获得三天内的天气预报。具体获取方式以及解析方法参见中国天气网提供的API文档。

12.3 程序开发

- **12.3.4 后台服务**
  - 最后，在AndroidManifest.xml文件中注册WeatherService，并声明连接互联网和发送SMS短信的两个用户许可

```
1    <service android:name=".Service.WeatherService"/>
2    <uses-permission android:name="android.permission.INTERNET" />
3    <uses-permission android:name="android.permission.SEND_SMS"/>
```

12.3 程序开发

- **12.3.5 用户界面**
  - 在用户界面设计上，采用可多分页快速切换的TabHost控件
  - WeatherDemo示例TabHost控件的每个标签页与一个Activity相关联，这样就可以将不同标签页的代码放在不同的文件中，而且每个标签页都可以有独立的选项菜单

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherDemo类是继承TabActivity的Tab标签页，共设置3个标签页
    - TAB1的标题为“天气预报”，关联的Activity为WeatherActivity
    - TAB2的标题为“历史数据”，关联Activity为HistoryActivity
    - TAB2的标题为“系统设置”，关联Activity为SetupActivity

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherDemo.java文件的完整代码如下:

```
1   package edu.hrbeu.WeatherDemo;
2
3   import android.app.TabActivity;
4   import android.content.Intent;
5   import android.os.Bundle;
6   import android.widget.TabHost;
7
8   public class WeatherDemo extends TabActivity {
9       @Override
10      public void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12
13          TabHost tabHost = getTabHost();
14          tabHost.addTab(tabHost.newTabSpec("TAB1")
```

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherDemo.java文件代码

```
15          .setIndicator("天气预报",
        getResources().getDrawable(R.drawable.tab_weather))
16          .setContent(new Intent(this, WeatherActivity.class)));
17
18          tabHost.addTab(tabHost.newTabSpec("TAB2")
19          .setIndicator("历史数据",getResources().getDrawable(R.drawable.tab_history))
20          .setContent(new Intent(this, HistoryActivity.class)));
21
22          tabHost.addTab(tabHost.newTabSpec("TAB3")
23          .setIndicator("系统设置",getResources().getDrawable(R.drawable.tab_setup))
24          .setContent(new Intent(this, SetupActivity.class)));
25      }
26  }
```

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherDemo.java中的代码只是用户界面的框架，设置了Tab标签页的图标、标题和所关联的Activity，标签页中的具体显示内容还要依赖于每个Activity所设置的界面布局
  - 界面布局包含
    - WeatherActivity
    - HistoryActivity
    - SetupActivity

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherActivity用户界面

12.3 程序开发

- **12.3.5 用户界面**
  - WeatherActivity在启动时并不能直接显示最新的天气信息，用户需要通过选项菜单的“启动服务”来开启后台服务，然后点击“刷新”获取最新的天气状况
  - 选项菜单还提供“停止服务”和“退出”选项
  - WeatherActivity使用的布局文件是tab_weather.xml，这是个较为繁琐的界面布局，多次使用了垂直和水平的线性布局
  - WeatherActivity的界面布局和代码并不难以理解，因此这里不再给出WeatherActivity.java和tab_weather.xml具体代码

12.3 程序开发

- **12.3.5 用户界面**
  - HistoryActivity主要用来显示SQLite数据库中的短信服务信息，显示的内容包括发送者的手机号码、时间和回复短信内容
  - 为了能够以列表的形式显示多行数据，并且定制每行数据的显示布局，这里使用了以往章节没有介绍过的ListActivity（Android.app.ListActivity）
  - 右图为HistoryActivity用户界面图

12.3 程序开发

- **12.3.5 用户界面**
  - ListActivity可以不通过setContentView()设置布局，也不必重载onCreate()函数，而直接将显示列表加载到ListActivity，增加了使用的便利性
  - 在WeatherDemo示例中，仍然使用setContentView()设置布局，这样做的好处是可以在界面中设置更为复杂的显示元素，例如在列表上方增加了提示信息“SQLite数据库中的短信服务信息”
  - 下方的代码是HistoryActivity.java文件的onCreate()函数中的设置布局和加载适配器的关键代码

```
1    setContentView(R.layout.tab_history);
2    setListAdapter(dataAdapter);
```

12.3 程序开发

• 12.3.5 用户界面

- tab_history.xml是HistoryActivity的布局文件， 下面先分析一下tab_history.xml的内容
- tab_history.xml文件的完整代码如下:

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      android:background="@drawable/black">
7
8      <TextView  android:layout_width="wrap_content"
9            android:layout_height="wrap_content"
10           android:text="SQLite数据库中的短信服务信息： ">
11      </TextView>
12      <ListView android:id="@android:id/list"
13         android:layout_width="fill_parent"
```

程序开发

- **12.3.5 用户界面**
  - tab_history.xml文件代码

```
14          android:layout_height="wrap_content"
15          android:layout_marginTop="2dip">
16      </ListView>
17  </LinearLayout>
```

  - tab_history.xml在代码的第12至第16行使用了ListView控件，并定义其系统ID值为“@android:id/list”，ListView的数据列配器是通过setListAdapter(dataAdapter)设置的

12.3 程序开发

• 12.3.5 用户界面

- ListView使用的是自定义布局，布局保存在data_row.xml文件中，data_row.xml的完整代码如下：

```
1   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2       android:orientation="horizontal"
3       android:layout_width="fill_parent"
4       android:layout_height="fill_parent"
5       android:background="@drawable/white"
6       android:layout_marginTop="2dip">
7
8       <LinearLayout android:orientation="vertical"
9           android:layout_width="fill_parent"
10          android:layout_height="fill_parent">
11
12          <TextView android:id="@+id/data_row_01"
13              android:layout_gravity="center_vertical"
14              android:layout_width="fill_parent"
```

12.3 程序开发

- **12.3.5 用户界面**
  - data_row.xml文件代码

```
15            android:layout_height="wrap_content"
16            android:textSize="12dip"
17            android:textColor="@drawable/black"/>
18
19        <TextView android:id="@+id/data_row_02"
20            android:layout_gravity="center_vertical"
21            android:layout_width="fill_parent"
22            android:layout_height="wrap_content"
23            android:textSize="12dip"
24            android:textColor="@drawable/black"
25            android:layout_marginTop="3dip"/>
26    </LinearLayout>
27  </LinearLayout>
```

12.3 程序开发

- **12.3.5 用户界面**
  - Android提供的数据适配器仅允许保存字符串数组或列表对象，如果希望使用自定义布局，则需要实现自定义的数据适配器，并继承Android提供的BaseAdapter（Android.widget.BaseAdapter）对象
  - 自定义的数据适配器在SmsAdapter.java文件中，其完整代码如下：

```
1  package edu.hrbeu.WeatherDemo.SMS;
2
3  import android.content.Context;
4  import android.view.LayoutInflater;
5  import android.view.View;
6  import android.view.ViewGroup;
7  import android.widget.BaseAdapter;
8  import android.widget.TextView;
9  import edu.hrbeu.WeatherDemo.DB.DBAdapter;
```

12.3 程序开发

- **12.3.5 用户界面**
  - SmsAdapter.java文件代码

```
10  import edu.hrbeu.WeatherDemo.R;
11
12
13  public class SmsAdapter extends BaseAdapter{
14      private LayoutInflater mInflater;
15      private static DBAdapter dbAdapter ;
16      private static SimpleSms[] smsList ;
17
18      public SmsAdapter(Context context){
19          mInflater = LayoutInflater.from(context);
20          dbAdapter = new DBAdapter(context);
21          dbAdapter.open();
22          smsList = dbAdapter.GetAllSms();
23      }
24
```

12.3 程序开发

- **12.3.5 用户界面**
  - SmsAdapter.java文件代码

```
25      public static void RefreshData(){
26          smsList = dbAdapter.GetAllSms();
27      }
28      @Override
29      public int getCount(){
30          if (smsList == null)
31              return 0;
32          else
33              return smsList.length;
34      }
35      @Override
36      public Object getItem(int position) {
37          if (smsList == null)
38              return 0;
39          else
```

12.3 程序开发

- **12.3.5 用户界面**
  - SmsAdapter.java文件代码

```
40              return smsList[position];
41          }
42          @Override
43          public long getItemId(int position) {
44              return position;
45          }
46
47          @Override
48          public View getView(int position, View convertView, ViewGroup parent)
        {
49              ViewHolder holder;
50              if(convertView == null){
51                  convertView = mInflater.inflate(R.layout.data_row, null);
52                  holder = new ViewHolder();
53                  holder.textRow01 = (TextView)
        convertView.findViewById(R.id.data_row_01);
54                  holder.textRow02 = (TextView)
        convertView.findViewById(R.id.data_row_02);
```

12.3 程序开发

- **12.3.5 用户界面**
  - SmsAdapter.java文件代码

```
55              convertView.setTag(holder);
56          }
57          else{
58              holder = (ViewHolder) convertView.getTag();
59          }
60
61          if (smsList != null){
62              String row01Msg ="（"+position+"）" +" 发送者： "+
        smsList[position].Sender+"， "+smsList[position].ReceiveTime;
63              holder.textRow01.setText(row01Msg);
64              holder.textRow02.setText(smsList[position].ReturnResult);
65          }
66          return convertView;
67      }
68
69      private class ViewHolder{
```

12.3 程序开发

• 12.3.5 用户界面

- SmsAdapter.java文件代码

```
70          TextView textRow01;
71          TextView textRow02;
72      }
73  }
```

- 继承BaseAdapter类要重载4个函数，包括getCount()、getItem()、getItemId()和getView()
- LayoutInflater是将XML文件中的布局映射为View对象的类，在代码第14行进行了声明，在代码第51行，将data_row.xml文件映射为View对象
- 代码第70行和第71行的内容，需要对应data_row.xml文件中的界面元素

12.3 程序开发

- **12.3.5 用户界面**
  - SetupActivity主要用来保存和恢复用户设置的运行参数
  - 第一次启动或恢复缺省设置（在选项菜单中）后，界面上会显示系统的缺省设置，包括城市名称、更新频率、是否提供短信服务、是否记录短信服务数据信息和短信服务的关键字
  - SetupActivity用户界面如右图所示：

12.3 程序开发

- **12.3.5 用户界面**
  - SetupActivity.java文件中，主要功能集中在RestoreDefaultSetup()、UpdateUI()和SaveConfig()三个函数上
  - RestoreDefaultSetup()用来恢复系统的缺省配置
  - UpdateUI()会根据保存在Config类中的数据更新SetupActivity的界面控件
  - SaveConfig()根据界面配置更改Config类，然后调用数据库适配器的DBAdapter.SaveConfig()函数，将Config类中的配置数据写入数据库

12.3 程序开发

- **12.3.5 用户界面**
  - RestoreDefaultSetup()、UpdateUI()和SaveConfig()函数代码如下：

```
1   private void RestoreDefaultSetup(){
2       Config.LoadDefaultConfig();
3       UpdateUI();
4       dbAdapter.SaveConfig();
5   }
6
7   private void UpdateUI(){
8       cityNameView.setText(Config.CityName);
9       refreshSpeedView.setText(Config.RefreshSpeed);
10       smsServiceView.setChecked(Config.ProvideSmsService.equals("true")?true:false);
11       saveSmsInfoView.setChecked(Config.SaveSmsInfo.equals("true")?true:false);
12       keyWorkView.setText(Config.KeyWord);
13   }
14
15   private void SaveConfig(){
```

12.3 程序开发

- **12.3.5 用户界面**
  - RestoreDefaultSetup()、UpdateUI()和SaveConfig()函数代码如下:

```
16      Config.CityName = cityNameView.getText().toString().trim();
17      Config.RefreshSpeed = refreshSpeedView.getText().toString();
18      if (smsServiceView.isChecked()){
19          Config.ProvideSmsService = "true";
20      }else{
21          Config.ProvideSmsService = "false";
22      }
23      if (saveSmsInfoView.isChecked()){
24          Config.SaveSmsInfo = "true";
25      }
26      else{
27          Config.SaveSmsInfo = "false";
28      }
29      Config.KeyWord = keyWorkView.getText().toString().trim();
30      dbAdapter.SaveConfig();
31  }
```

12.3 程序开发

- **12.3.5 用户界面**
  - 为了所有定义的Activity和ListActivity生效，在AndroidManifest.xml文件中注册所有定义的组件

```
<activity android:name=".WeatherDemo"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".WeatherActivity"/>
<activity android:name=".HistoryActivity"/>
 <activity android:name=".SetupActivity"/>
```

习题：

1.本章综合示例使用TabHost和TabActivity实现Tab导航栏，尝试使用操作栏和Fragment实现该示例。

THANKS

谢　　谢　　观　　看