

长沙民政职业技术学院

毕业设计说明书

题目:基于 Android 的天气系统的设计与实现

类型:

产品设计	工艺设计	方案设计
√		

学生姓名: 沈程璞

学 号: 1425113301

学 院: 软件学院

专 业: 软件技术（移动应用开发方向）

班 级: 移动 1433

学校指导教师: 李政仪

企业指导教师:

2016 年 10 月 30 日

摘 要

20 世纪起，人类气象学的发展十分迅速，在 1853~1856 年期间，为了去争夺巴尔干半岛，俄罗斯和英法两个国家爆发了克里木战争，结果俄罗斯战败，正是这次的战争，诞生了天气预报。具体的事情经过就不做说明了，但这充分说明了天气预报的重要性，随着人类气象学的进一步发展，学术理论的进一步丰富，以及卫星的出现，对于天气的分析预报越来越准确。到如今，随着智能手机的大规模普及，获取天气预报已经成为人们生活中必不可少的一部分了！天气预报融入了人们的生活，我们每天总是习惯性的通过获取天气预报，来判断出行是否需要携带雨伞，添加衣物，甚至是否合适劳动……，总之现代人类的方方面面已经完全离不开天气预报了！

正是基于这个原因，萌生了自己设计一个天气预报的想法，基于 Android 平台，使用中国天气网数据接口，实现全国各城市天气查询！

【关键词】：Android；中国天气网

目录

摘 要.....	2
1. 绪 论.....	5
1.1 选题原因.....	5
2. 相关技术综述.....	6
2.1 ANDROID 平台.....	6
2.2 ANDROID 系统架构.....	6
2.3 应用组件.....	9
活动.....	10
服务.....	10
广播接收器.....	11
内容提供.....	11
2.4 JSON 数据格式.....	11
3. 安卓天气系统需求分析与设计.....	14
3.1 系统需求分析.....	14
用户群体:	14
系统需求详细说明:	14
3.2 功能模块设计.....	15
城市选择.....	16
显示天气.....	16
切换城市.....	17
更新天气.....	17
4. 安卓天气系统的实现与测试.....	18
4.1 创建数据库和表.....	18
保存城市对象数据到数据库.....	19
获取所有城市.....	19
根据名称匹配一个或多个城市.....	19
将状态更新为已有数据.....	20
4.2 遍历全国城市数据.....	20
4.3 显示天气信息.....	27



4.4 切换城市和手动更新天气.....	32
4.5 后台自动更新天气.....	36
5.结论.....	39
致谢.....	42
参考文献.....	42

1. 绪 论

1.1 选题原因

在以前我们获取天气预报的方式往往是通过电视。自从人类诞生以来，我想天气是人们不得不考虑的问题，随着人类活动范围和领域越来越广，天气变得越来越重要，当然专业领域，对我们来说相隔太遥远，但是，对于我们普通人来讲天气依然是重要的。不论是下地干活，还是出门上班、外出旅游，我们都需要提前获取天气，做好相应准备。所以说能够及时获取天气预报十分重要。

过去我们获取天气最常见的便是通过电视，但是电视有什么局限性呢？我们知道，天气是多变的，尤其是夏天，很可能随时下雨，而电视只是对第二天的天气做一个整体预测，所以在很多时候人们常常抱怨天气不准。这还仅仅是其中一个缺点，电视往往是固定播出时间的，错过天气预报那是很常见的事，更不利的是，由于中国幅员辽阔，某个电视台不可能播报全国所有城市的天气预报，这在地域方面的误差就变得更大了，因为东边下雨西边太阳也不是罕见的事！并且想要获得两个不同地方的天气预报变得十分不方便！

直到 2007 年，乔布斯发布了第一代苹果手机，但是苹果对于大多数的中国人来讲，是没有那个消费能力的，直到几年后，安卓机迅猛发展，因其相对低廉的价格，在中国快速普及，也就是这几年移动互联网取得高速发展，现在几乎是人手一台智能手机，它的出现也改变了我们的生活，生活化得来讲，我们可以通过手机购物、支付、叫外卖...当然也包括查看天气预报！相对于传统电视播报天气预报，手机可以实时查看天气，自由更换城市，不受时间地域限制，天气更新及时方便，相对传统电视，可以获取更丰富的天气信息，比如 pm2.5，具体时间段天气以及气温变化，紫外线照射等等，尤其是近几年雾霾引起人们重视！

2.相关技术综述

2.1 Android 平台

Android 是一种基于 Linux 的开源操作系统，主要用于移动设备，例如智能手机和平板电脑，由 Google 公司和开放手机联盟领导及开发。Android 最初是由 Andy Rubin 开发，主要支持手机。2005 年 8 月被 Google 收购。2007 年 11 月份，Google 和 84 家硬件公司、软件开发公司以及电信营运商组建开放手机联盟共同研发改良 Android 系统。之后 Google 以 Apache 开源许可证的授权方式，发布了 Android 的源代码。第一部手机在 2008 年 10 月正式发布，Android 逐渐扩展到平板电脑及其他领域上，如电视、数码相机、游戏机等。2011 年第一季度，Android 在全球的市场份额第一次超过塞班系统，跃升为世界第一。2013 年的第四季度，Android 平台手机的全球市场份额已经达到 78.1%。2013 年 09 月 24 日 Android 迎来了 5 周岁生日，全世界采用该系统的设备数量已经达到 10 亿台之多。

2.2 Android 系统架构

Android 是基于 linux 内核的移动设备开发平台和操作系统，主要分为 4 层，依次是应用程序层，应用框架层，系统运行库层和 linux 内核层。如图 2-1 所示。

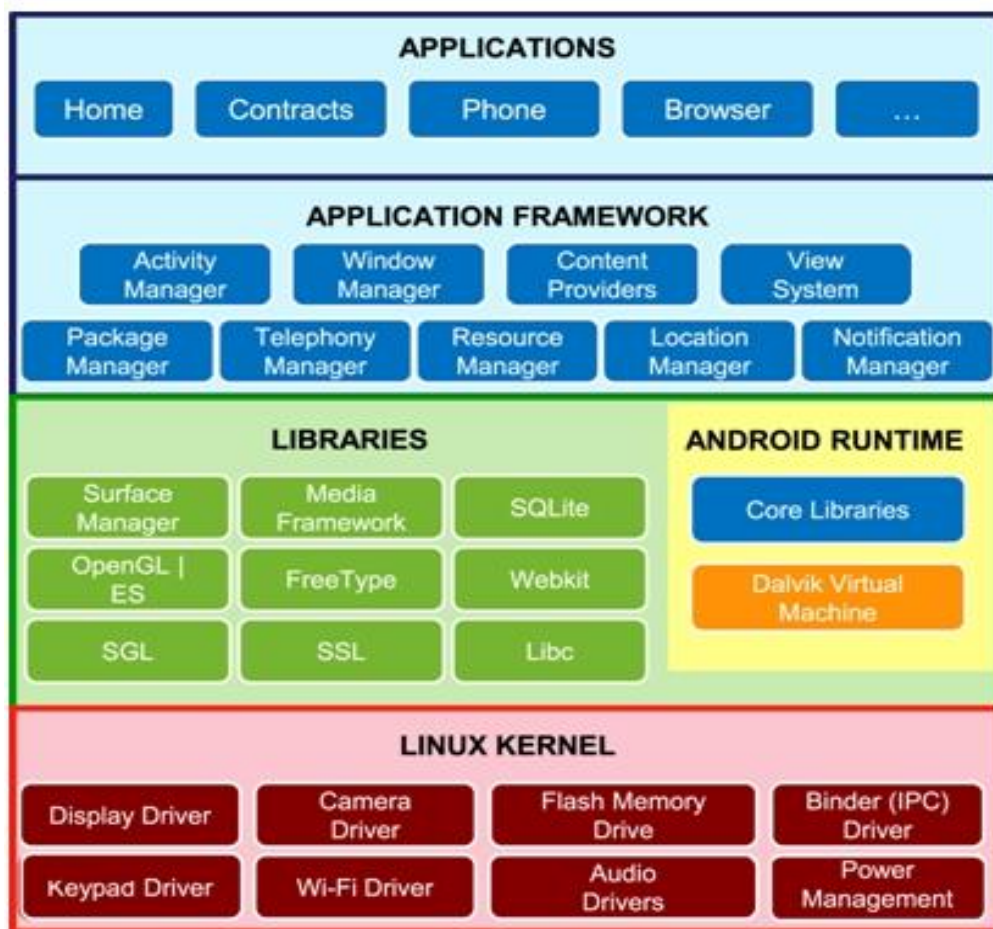


图 2-1 Android 系统架构图

下面分别讲解一下各个部分：

1. 程序应用

Android 连同一个核心应用程序包一起发布，这个应用程序包货了邮件客户端、短信客户端、日历、联系人管理、地图、浏览器等等。所有这些程序都是用 Java 编写出来的。

2. 应用程序框架

应用程序框架是一个程序的核心，所有参与开发的程序员都要遵守这个规定，大家在遵守规定的基础上进行扩展，但程序的主体结构始终是保持一致的。它的作用就是让程序保持清晰并且一目了然，满足不同的需求又不相互影响。开发应用时都是通过框架来和安卓的底层进行交互的，而接触最多的就是应用程序框架层。Android 分为应用层、系统运行库层、应用框架层和 Linux 内核层。

Android 系统提供给应用开发者的本身就是一个框架，所有的应用开发都必须遵守这个框架的原则。我们在开发应用时就是在这个框架上进行扩展。Android 应用框架功能如下。

*android.app:提供高层的程序模型和基本的运行环境。

*android.content:包含对各种设备上的数据进行访问和发布。

*android.database:通过内容提供者浏览和操作数据库。

*android.graphics:底层的图形库，包含画布、颜色过滤、点、矩形，可以将它们直接绘制到屏幕上。

*android.location:定位和相关服务的类。

*android.media:提供一些类管理多种音频、视频的媒体接口。

android.net:提供帮助网络访问的类，超过通常的 java.net. 接口。

*android.os:提供了系统服务、消息传输和 IPC 机制。

*android.opengl:提供 OpenGL 的工具。

*android.provider:提供访问 Android 内容提供者的类。

*android.telephony:提供与拨打电话相关的 API 交互。

*android.view:提供基础的用户界面接口框架。

*android.util:涉及工具性的方法，例如时间日期的操作。

*android.webkit:默认浏览器操作接口。

*android.widget:包含各种 UI 元素(大部分是可见的)在应用程序的布局中。

3. Android 程序库

Android 包括一个被 Android 系统中各种不同组件所使用的 C/C++ 库。该库通过 Android 应用程序框架为开发者提供服务。

以下是一些主要的核心库：

1) 系统 C 库：一个从 BSD 继承来的标准 C 系统函数库 (libc)，专门为基于

Embedded Linux 的设备定制。

2) 媒体库: 基于 PacketVideo OpenCORE; 该库支持录放, 并且可以录制许多流行的音频视频格式, 还有静态映像文件包括 MPEG4、H.264、MP3、AAC、JPG、PNG。

3) Surface Manager: 对显示子系统的管理, 并且为多个应用程序提供 2D 和 3D 图层的无缝融合。

4) LibWebCore: 一个最新的 Web 浏览器引擎, 用来支持 Android 浏览器和一个可嵌入的 Web 视图。

5) SGL: 一个内置的 2D 图形引擎。

6) 3D libraries: 基于 OpenGL ES 1.0 APIs 实现; 该库可以使用硬件 3D 加速(如果可用) 或者使用高度优化的 3D 软加速。

7) FreeType: 位图(bitmap)和向量(vector)字体显示。

8) SQLite: 一个对于所有应用程序可用、功能强劲的轻型关系型数据库引擎。

4. Android 运行库

安卓包括了一个核心库, 它提供了 Java 编程语言核心库的绝大部分功能

每个安卓程序都在自己的进程中运行, 它们都拥有一个独立的 Dalvik 虚拟机实例, Dalvik 是针对同时高效地运行 VMs 而实现的。该虚拟机.dex 文件, 该格式文件针对最小内使用情况做了特定优化。Dalvik 虚拟机是基于寄存器的, 所有的类都 Java 汇编器编译的, 然后通过 SDK 中的 DX 工具转换.dex 格式, 在交给虚拟机来运行。

Dalvik 还依赖于 Linux 的一些功能, 录入线程机制和内存管理机制。

5. Linux 内核

安卓的核心系统服务仍是依赖于 Linux 内核, 比如内存管理、安全性, 网络协议、驱动模型和进程管理等。Linux 内核同时作为硬软件之间的硬件抽象层。

2.3 应用组件

Android 开发四大组件分别是: 活动 (Activity): 用于表现功能。服务 (Service): 后台运行服务, 不提供界面呈现。广播接收器 (BroadcastReceiver): 用于接收广播。内容提供商 (Content Provider): 支持在多个应用中存储和读取数据, 相当于数据库。

活动

在安卓中，所有程序都运行在 Activity 之中，在安卓程序中，一般一个活动代表一屏。如果把它比作浏览器，那它就像是一个网页。在 Activity 中可以添加一些空间，比如按钮，文本框等等。一般一个程序是由很多个 Activity 组成的，在我们平时使用的时候可以看到，并且是可以相互跳转的，比如我们按下一个按钮就跳转到另一个页面去了，与网页跳转不同的是，Activity 之间跳转是可能带有返回值的，这么做在很多时候非常方便。

当我们打开一个新的界面时，前一个会被设置为暂停，并且压到历史堆栈里面。而回退则可以返回到以前打开过的界面。在这里我们其实可以选择移除一些不需要的屏幕，因为安卓会把每个应用的开始到当前的每个界面都保存在堆栈中。

服务

Service 是 android 系统中的一种组件，

它其实和 Activity 级别差不多，不过不可以在前台运行，只能在后台运行，也就是说用户是看不见的，但它同样可以和其它组件进行交互，服务可以运行很久，不过没有界面。这么说似乎有些抽象的样子，下面举个例子：我们都有过边听音乐边上网的经历吧？是的，我们在上网干其它事情的时候音乐并没有停止，这是什么原因呢？原来音乐播放就是通过 Service 来控制的。其实服务的用处远不止如此，比如检测 SD 卡文件变化，获取位置信息等等，我们在平时使用手机，细心地话就不难发现，这就是服务！

开启 Service 有两种方式：

(1) Context.startService ()：Service 会经历 onCreate -> onStart (如果 Service 还没有运行，则 android 先调用 onCreate () 然后调用 onStart ()；如果 Service 已经运行，则只调用 onStart(), 所以一个 Service 的 onStart 方法可能会重复调用多次)；StopService 的时候直接 onDestroy, 如果是调用者自己直接退出而没有调用 StopService 的话, Service 会一直在后台运行。该 Service 的调用者再启动起来后可以通过 stopService 关闭 Service。注意，多次调用 Context.startservice () 不会嵌套 (即使会有相应的 onStart () 方法被调用)，所以无论同一个服务被启动了多少次，一旦调用 Context.stopService () 或者 StopSelf ()，他都会被停止。补充说明：传递给 StartService () 的 Intent 对象会传递给 onStart () 方法。调用顺序为：onCreate --> onStart (可多次调用) --> onDestroy。

(2) `Context.bindService()`: `Service` 会经历 `onCreate()` --> `onBind()`, `onBind` 将返回给客户端一个 `IBind` 接口实例, `IBind` 允许客户端回调服务的方法, 比如得到 `Service` 运行的状态或其他操作。这个时候把调用者 (`Context`, 例如 `Activity`) 会和 `Service` 绑定在一起, `Context` 退出了, `Service` 就会调用 `onUnbind` --> `onDestroyed` 相应退出, 所谓绑定在一起就共存亡了。

广播接收器

在安卓中, 广播接收器是一种广泛用在程序之间传输信息的机制, `BroadcastReceiver` 就是对发出来的广播进行过滤并且响应的一类组件。例如, 当一个电话打进来或者接受到一条短信甚至程序下载完成的时候, 都可以用它进行处理。它不可以生成 UI, 所以和服务一样, 用户也是看不到的。但是可以通过通知来告诉用户这些事件发生了。广播可以分为静态注册和动态注册, 只要注册了, 就算程序煤油运行, 系统也会在需要的时候启动, 应用之间还可以通过 `Context.sendBroadcast()` 把自己的广播给其他程序, 这在我们使用的过程中经常发现, 一个应用唤醒另一个应用, 就是通过广播来实现的。

内容提供

在安卓里面, 对数据的保护程度还是很高的, 除了放在内存卡中的数据之外, 一个应用的其他所有数据都是不允许其他程序直接访问的。但是很显然, 如果这么封闭, 那么灵活性也太低了, 应用体验一定不好, 所以内容提供者就是解决这个问题的, 它可以通过派生 `Content Provider` 类, 封装成一枚 `Content Provider`, 每个 `Content Provider` 都用一个 `uri` 作为独立的标识, 就像一个记号一样。它有两种类型, 一种是带列表的, 一种是带 `id` 的, 反正只要调用者清除, 就没问题!

2.4 JSON 数据格式

`JSON(JavaScript Object Notation)` 是一种非常轻量级的文本数据交换语言, 以文本为基础, 很适合阅读。虽然说 `JSON` 是 `Javascript` 的一个子集, 但这并不代表它不能用在别的语言中。它是一种标准, 只要符合 `JSON` 的语法构造规则、任何语言都可以使用和解析 `JSON` 这种数据格式。

`JSON` 有两种数据结构: `object` (对象) 和 `array` (数组)。对象就是一个“名称-值”

对的集合，一个对象一“{”开始，“}”结束，名称和价值之间映射冒号隔开，“名称-值”对之间使用逗号分隔。如图 2-2 所示：

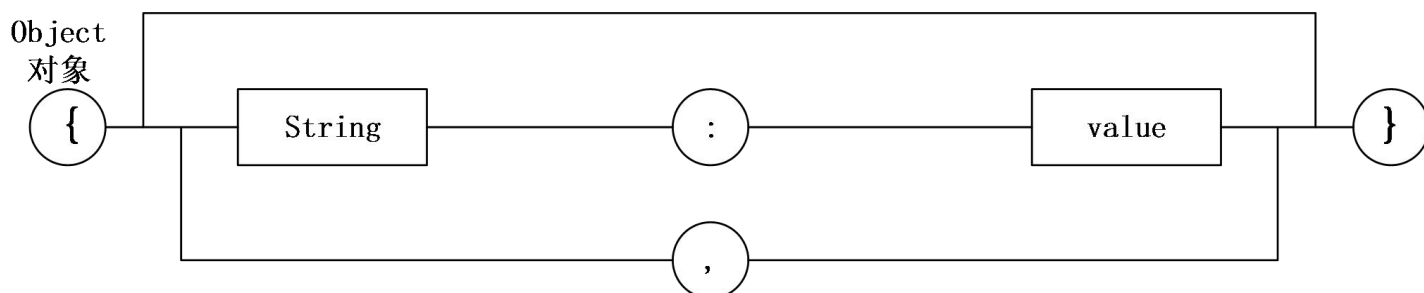


图 2-2 JSON 对象结构图

数组是值（value）的有序组合。一个以“[”开始，以“]”结束，值与值之间使用逗号隔开。如图 2-3 所示：

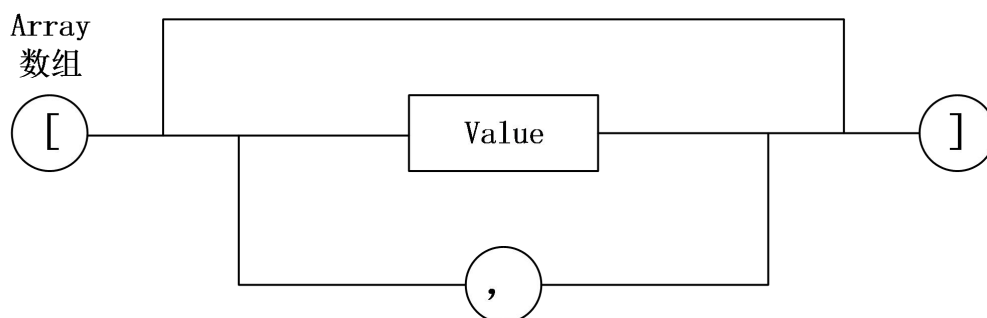


图 2-3 JSON 数组结构图

值（value）可以是字符串（string）、数值（number）、true、false、null、对象（object）和数组（array）。并且这些结构可以嵌套使用。

JSON 和 XML 一样都是数据交换格式，但是 XML 是一个完整的标记语言、JSON 不是，这使得 XML 在程序判读方面要花费很多的时间。相比较的话，可以发现它们的异同点如下：

- 它们都有很好的可读性和丰富的解析手段。
- 在扩张性方面、XML 由于本身的结构，有很好的扩展性，JSON 扩张性不如 XML。
- JSON 编码难度和解码难度要比 XML 容易很多。
- JSON 相对于 XML 而言，数据体积更小。
- JSON 对数据的描述比 XML 要差。
- JSON 的速度比 XML 快很多。

不同的场合适合用不同的数据格式，XML 在数据存储、扩展和高级检索方面有自己的优势，但 JSON 由于比 XML 体积小很多、能够支持快速解析、很适合应用在互联网数据传输上。

Android 系统提供了 JSON.org 包来对 JSON 数据进行操作，这个包主要包括以下几个类：JSONObject 类是构造 JSON 对象的类，JSONArray 类是构造 JSON 数组的类，根据上面说的 JSON 语法规则，使用两个类就可以构造 JSON 对象了。JSONStringer 类是使用另一种方式构造一个 JSON 对象，JSONTokener 类是对 JSON 数据进行解析。JSONException 类封装了一些常见的异常处理。在 Android 系统开发中，开发者使用这些类就可以对 JSON 进行基本的操作了。

3. 安卓天气系统需求分析与设计

3.1 系统需求分析

做一款软件，当然是根据需求去做设计，用户需要什么样的功能，用户的主要群体是什么，通过调研进行具体分析，得到相应结论，为后面的软件设计做指导。

用户群体：

一款天气应用，其实几乎对每个人都是适用的，前面已经做过说明，但是随着智能手机的进一步普及，很多老年人也开始使用，那么简洁、易用应当成为软件设计的重要参考！

系统需求详细说明：

作为一款天气应用，至少应该具备以下功能：

1. 可以罗列出全国所有的城市。
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。

看起来似乎只有简单的四个功能点，但是却涉及了网络、定位、数据存储、服务等技术！很显然，查看天气首先需要选择城市，并且任意更换城市，而天气是实时变化更新的，那么自然也需要提供更新功能，如果每次都需要手动更新，那么是非常不人性化的，为了更好的用户体验，将提供手动、自动更新两种方式！用户需求功能如图3-1所示：

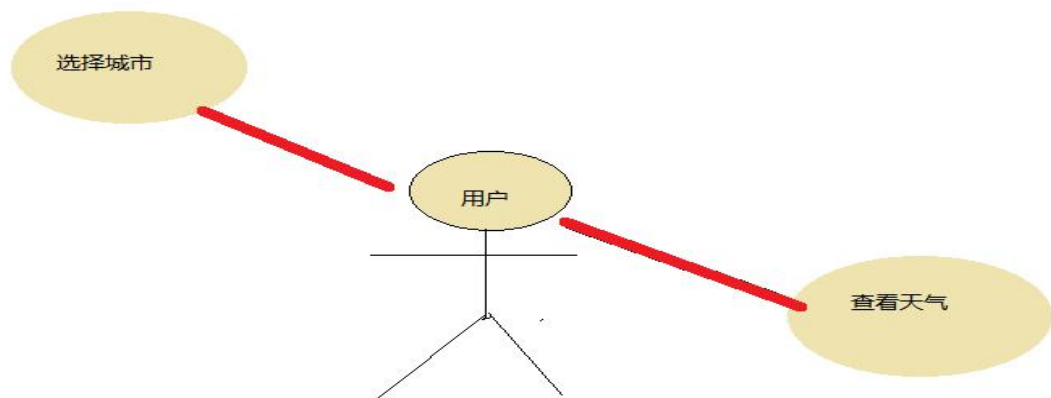


图 3-1 用户需求示意图

3.2 功能模块设计

通过对需求分析，之后就是需要将这些需求转换成具体的功能模块设计！前面已经分析过，作为天气应用，覆盖的群体十分广泛，应当尽可能地简洁易用，本着这个原则来进行设计！功能模块如图 3-2 所示：

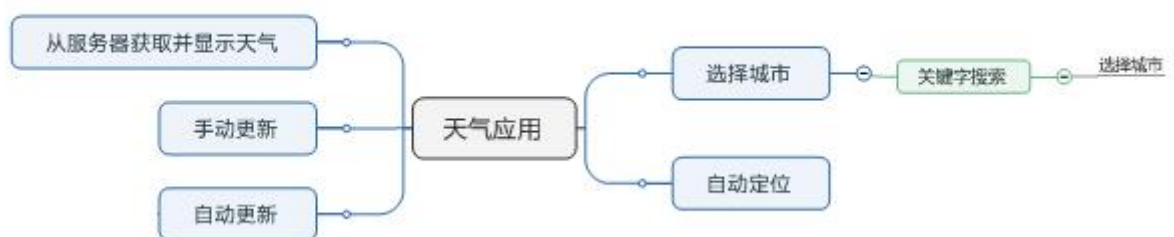


图 3-2 功能模块示意图

城市选择

要想查看天气，首先需要知道你的位置，在传统电视获取天气信息的时候，为了获取更精确的天气情况，大多数情况下总是选择收看省级卫视，甚至地方电视台的天气预报，为的就是为了获取更准确的天气预报，因为很多时候天气在很小的范围之内都是有差别的，所以我们获取天气的第一步就是选择地点，下面通过一个简单的示意图来演示，如图 3-3 所示

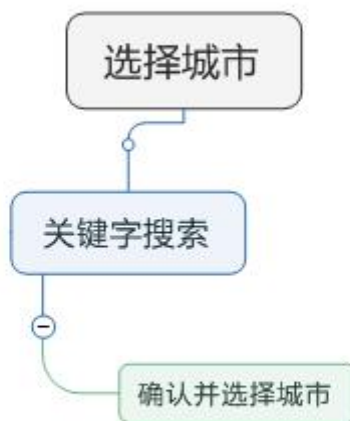


图 3-3 地理位置选择示意图

显示天气

以前我们通过电视看天气预报，会听到播报员说，xx 地 xx 天气转 xx 天气，xx 摄氏度到 xx 摄氏度！很显然，这些都是必不可少的天气信息。用户可以非常直观的看到这些天气信息！在传统电视播报天气预报，因为固定的时间，且多变的天气，很多时候由

于人类活动会忘记天气信息，而电视不会一直播放天气预报，不过手机查看则不存在这个问题，它可以在任何时候查看天气，即使有一段时间没有网络，也会保存最近一次联网的天气数据，准确度还是比较高的！

切换城市

前面已经说到，移动平台查看天气的优点之一便是可以随意更改地点，查看不同地方的天气，在以前要想虎丘区不同地方的天气，一种办法是收看央视的天气预报，但是只能获得各省省会天气预报，这样的天气准确度会大大降低，还有一种办法是收看其它地方的地方台或卫视，但是大家也知道收看其他省份的地方台基本是不具备这个条件的，而收看其它卫视因为不熟悉播出时间，或者播出时间重合，这是十分常见的事，这对获取其它地方天气是极为不方便的！

更新天气

天气是实时变化的，这个大家都清楚，在传统电视，我们一天只能查看一次天气预报，对于天气变化不大的季节还是没有什么影响的，但是对于多变的天气就有些力不从心了，而手机就可以在很大程度上解决这个问题，更新天气有手动和自动两种方式，自动更新通过程序的定时任务会每隔一段时间自动从服务器获取数据，而手动更新则通过用户点击更新按钮实时更新天气。这样可以最大程度获取更准确的信息！

4. 安卓天气系统的实现与测试

4.1 创建数据库和表

一个天气应用，要想实现查看天气，首先便要实现对位置的选择，我们实现天气的精确范围是县一级，由于采用和风天气接口，直接提供全国县级城市数据，显然我们不可能一个个去找，全国这么多县，所以最后采用搜索方式进行选择城市！为了项目能有更好的结构，包结构如下：







 activity	2016/11/9 8:23	文件夹
 db	2016/11/9 8:23	文件夹
 model	2016/11/9 8:23	文件夹
 receiver	2016/11/9 8:23	文件夹
 service	2016/11/9 8:23	文件夹
 util	2016/11/9 8:23	文件夹

图 4-1 包结构示意图

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。包建好了，首先便是要创建数据库和表，这样才能将服务器数据存储到本地

```
public class MaoWeatherOpenHelper extends SQLiteOpenHelper {  
  
    //创建城市表  
    private static final String CREATE_CITY = "CREATE TABLE CITY(ID INTEGER PRIMARY  
KEY," +  
        "CITY_NAME_EN TEXT,CITY_NAME_CH TEXT,CITY_CODE TEXT)";  
    //创建有无数据状态表  
    private static final String DATA_STATE = "CREATE TABLE DATA_STATE(STATE  
INTEGER PRIMARY KEY)";  
    //更新状态表数据为0表示暂无数据  
    private static final String INSERT_DATA_STATE = "INSERT INTO DATA_STATE  
VALUES(0)";  
}
```



```
public MaoWeatherOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory
factory, int version) {

    super(context, name, factory, version);

}

@Override
public void onCreate(SQLiteDatabase db) {

    db.execSQL(CREATE_CITY);
    db.execSQL(DATA_STATE);
    db.execSQL(INSERT_DATA_STATE);

}

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

}

}
```

上面代码先是创建表，然后在onCreate里执行，为了后面更好的开发，在model包下再创建一个实体类City，这个类很简单，就是生成数据表对应字段的set、get方法，接下来就是创建数据操作类，将常用的数据库操作封装

保存城市对象数据到数据库

此方法把获取到的城市对象保存到数据库中，方便后面直接从本地数据库查询查询匹配城市名称，而不用再去从网络上获取！

获取所有城市

从数据库取得数据并添加到动态数组里

根据名称匹配一个或多个城市

根据用户输入的名称进行匹配，取出相关的城市

将状态更新为已有数据

将数据的状态更新为已有数据，避免再次从网络获取数据。

在数据库操作类中写出了四个访法，用以保存、读取、更新城市数据和状态，这几个方法将为后面的数据库操作提供极大便利！

4.2 遍历全国城市数据

全国所有城市的数据都是从服务器上获取的，所以是肯定需要和数据库进行交互的，因此在 util 包下创建 HttpUtil 类

```
public class HttpUtil {  
    public static void sendHttpRequest(final String address, final HttpCallback callback) {  
  
        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 inputStream = connection.getInputStream();  
  
                    BufferedReader reader = new BufferedReader(new  
InputStreamReader(inputStream));  
                    StringBuilder builder = new StringBuilder();  
                    String line = "";  
  
                    while ((line = reader.readLine()) != null) {
```

```
        builder.append(line);
    }

    if (callback != null) {
        callback.onFinish(builder.toString());
    }

    } catch (Exception e) {
        if (callback != null)
            callback.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 HttpCallback {

    void onFinish(String response);

    void onError(Exception e);

}
```

在Android 上发送HTTP 请求的方式一般有两种，URLConnection 和HttpClient，这里使用的是 HttpURLConnection 方法。首先需要获取到URLConnection 的实例，一般只需new 出一个URL 对象，并传入目标的网络地址，然后调用一下openConnection() 方法即可，如下所示：

```
URL url = new URL("http://www.baidu.com");
```

```
URLConnection connection = (URLConnection) url.openConnection();
```

得到了URLConnection 的实例之后，我们可以设置一下HTTP 请求所使用的方

法。常用的方法主要有两个，GET 和POST。GET 表示希望从服务器那里获取数据，而POST 则表示希望提交数据给服务器。写法如下：

```
connection.setRequestMethod("GET");
```

接下来就可以进行一些自由的定制了，比如设置连接超时、读取超时的毫秒数，以及服务器希望得到的一些消息头等。这部分内容根据自己的实际情况进行编写，示例写法如下：

```
connection.setConnectTimeout(8000);
```

```
connection.setReadTimeout(8000);
```

之后再调用getInputStream()方法就可以获取到服务器返回的输入流了，剩下的任务就是对输入流进行读取，如下所示：

```
InputStream in = connection.getInputStream();
```

最后可以调用disconnect()方法将这个HTTP 连接关闭掉，如下所示：

```
connection.disconnect();
```

在上面，在 sendHttpRequest 方法中开启了子线程，使用 HttpURLConnection 发送了一条 HTTP 请求，接着用 BufferedReader 对服务器返回的流进行读取，并进行判断，由于服务器返回的是 JSON 数据，接下来创建一个工具类—Utility 解析和处理数据

```
public class Utility {  
  
    //处理从服务器获取的城市数据  
  
    public synchronized static boolean handleCityResponse(MaoWeatherDB maoWeatherDB,  
String response) {  
  
        if (!TextUtils.isEmpty(response)) {  
            try {  
                JSONArray jsonArray = new JSONObject(response).getJSONArray("city_info");  
                for (int i = 0; i < jsonArray.length(); i++) {  
                    JSONObject city_info = jsonArray.getJSONObject(i);  
                    City city = new City();  
                    String city_name_ch = city_info.getString("city");  
                    String city_name_en = "";  
                    String city_code = city_info.getString("id");  
                    city.setCity_code(city_code);  
                    city.setCity_name_en(city_name_en);  
                }  
            }  
        }  
    }  
}
```



```
        city.setCity_name_ch(city_name_ch);
        maoWeatherDB.saveCity(city);
    }
} catch (Exception e) {
    e.printStackTrace();
}
return true;
}
return false;
}
}
```

上面将城市信息存储到数据库 接下来在res/layout 目录中新建choose_area.xml 布局，添加一些控件进行操作！接下来就是用于遍历城市数据的活动了

```
public class CityChooseActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_citychoose);

        mMaoWeatherDB = MaoWeatherDB.getInstance(this); //获取数据库处理对象
        mSharedPreferences = PreferenceManager.getDefaultSharedPreferences(this); //获取本地
        存储对象
        mEditor = mSharedPreferences.edit(); //获取本地存储对象

        //先检查本地是否已同步过城市数据，如果没有，则从服务器同步
        if (mMaoWeatherDB.checkDataState() == NONE_DATA) {
            queryCitiesFromServer();
        }

        mCities = queryCitiesFromLocal(""); //获取本地存储的所有的城市
    }
}
```



```
//搜索框，设置文本变化监听器
editText = (EditText) findViewById(R.id.edit_city);
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {

        mCities = queryCitiesFromLocal(s.toString());//每次文本变化就去本地数据库
        查询匹配的城市

        mAdapter.notifyDataSetChanged();//通知更新
    }

    @Override
    public void afterTextChanged(Editable s) {
    }
});

mAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
cityNames);//适配器初始化
mListView = (ListView) findViewById(R.id.list_view_cities);
mListView.setAdapter(mAdapter);

//ListView的Item点击事件
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        mCity_selected = mCities.get(position);//根据点击的位置获取对应的City对象
        queryWeatherFromServer();//根据点击的城市从服务器获取天气数据
    }
});
```




```
}

//从服务器取出所有的城市信息

private void queryCitiesFromServer() {

    String address = " https://api.heweather.com/x3/citylist?search=allchina&key=" +
MaoWeatherActivity.WEATHER_KEY;

    showProgressDialog();

    HttpUtil.sendHttpRequest(address, new HttpCallback() {

        @Override

        public void onFinish(String response) {

            if (Utility.handleCityResponse(mMaoWeatherDB, response)) {

                runOnUiThread(new Runnable() {

                    @Override

                    public void run() {

                        closeProgressDialog();

                        mMaoWeatherDB.updateDataState();

                    }

                });

            }

        }

        @Override

        public void onError(final Exception e) {

            runOnUiThread(new Runnable() {

                @Override

                public void run() {

                    closeProgressDialog();

                    Toast.makeText(CityChooseActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();

                }

            });

        }

    })

}
```



```
    });  
}  
  
//从本地数据库取出相似的城市名称  
private List<City> queryCitiesFromLocal(String name) {  
    List<City> cities = mMaoWeatherDB.loadCitiesByName(name);  
    cityNames.clear();  
    for (City city : cities) {  
        cityNames.add(city.getCity_name_ch());  
    }  
    return cities;  
}  
  
//从服务器获取天气数据  
private void queryWeatherFromServer() {  
  
    String address = "https://api.heweather.com/x3/weather?cityid=" +  
mCity_selected.getCity_code() + "&key=" + MaoWeatherActivity.WEATHER_KEY;  
    showProgressDialog();  
  
    HttpUtil.sendHttpRequest(address, new HttpCallback() {  
        @Override  
        public void onFinish(String response) {  
            //将从服务器获取的JSON数据进行解析  
            if (Utility.handleWeatherResponse(mEditor, response)) {  
                //注意这里对线程的处理  
                runOnUiThread(new Runnable() {  
                    @Override  
                    public void run() {  
                        closeProgressDialog();  
                        //处理完天气数据,说明已经保存到本地,我们不用再把数据封装到Intent里面返回给MaoWeatherActivity  
                        //可以在onActivityResult里面从本地存储中获取
```

```

        setResult(RESULT_OK);
        finish();
    }
});
}

@Override
public void onError(Exception e) {
    //注意这里对线程的处理
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            closeProgressDialog();
            Toast.makeText(CityChooseActivity.this, "数据同步失败",
Toast.LENGTH_SHORT).show();
        }
    });
}
});
}

```

在onCreate（）方法中，如果读取到数据就进行加载，否则从服务器进行读取，即调用queryCitiesFromServer（）方法，该方法会对传入的参数进行拼装查询地址，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handleWeatherResponse()方法来解析和处理服务器返回的数据，并存储到数据库中。

4.3 显示天气信息

接下来就是要显示天气信息了，要显示天气信息，自然需要一个新的页面，因此先

创建一个布局文件，用来显示天气信息在res/layout 目录中新建weather_layout.xml，用以显示天气信息，进行相关操作！这个界面十分简单，就是利用一些TextView控件来显示天气信息，接下来在Utility类中，还需要添加方法，解析和处理服务器返回数据，此方法将JSON数据的天气信息其全部解析出来，并用SharedPreferences存储起来,接下来应当创建活动显示天气信息了，在activity包下创建MaoWeatherActivity类，代码如下：

```
public class MaoWeatherActivity extends Activity {

    public static final String WEATHER_KEY = "735a9b1324f04671a3e3f27c330dd9b4 ";

    private City mCity_current = new City();//当前显示的城市对象

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.activity_maowweather);

        //实例化本地存储

        mSharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);

        mEditor = mSharedPreferences.edit();

        //变更城市（小房子按钮）

        mChangeCityButton = (Button) findViewById(R.id.button_changeCity);

        mChangeCityButton.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                //就是启动CityChooseActivity

                Intent intent = new Intent(MaoWeatherActivity.this, CityChooseActivity.class);

                //以要求返回结果的方式启动

                startActivityForResult(intent, REQUEST_CODE);

            }

        });

    }

}
```



//刷新按钮

```
mRefreshButton = (Button) findViewById(R.id.button_refresh);
```

```
mRefreshButton.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

```
    public void onClick(View v) {
```

```
        //从服务器更新
```

```
        updateWeatherFromServer();
```

```
    }
```

```
});
```

//这个是为了在第一次安装的时候，判断本地存储还没有数据，所以默认获取北京的数据

//如果需要修改，可以从和风天气官网 <http://www.heweather.com/documents/cn-city-list>查询城市

ID

```
if (mSharedPreferences.getString("city_code", null) == null) {
```

```
    mCity_current.setCity_code("CN101010100");
```

```
    updateWeatherFromServer();
```

```
    } else {
```

```
        //有数据，则从本地取出来，也就是上次访问的城市，先确定这个
```

```
        loadWeatherData(mSharedPreferences.getString("city_code", null),
```

```
mSharedPreferences.getString("city_name_ch", null), mSharedPreferences.getString("update_time", null),
```

```
mSharedPreferences.getString("data_now", null), mSharedPreferences.getString("txt_d", null),
```

```
mSharedPreferences.getString("txt_n", null), mSharedPreferences.getString("tmp_min", null),
```

```
mSharedPreferences.getString("tmp_max", null));
```

```
        //然后从服务器更新一次
```

```
        updateWeatherFromServer();//可以注释掉，使用服务进行自动更新
```

```
    }
```

```
//启动自动更新服务
```

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

```
startService(intent);
```

```
LinearLayout linearLayout = (LinearLayout) findViewById(R.id.adLayout);
```

```
//linearLayout.addView(adView);
```

```
}
```

```
@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    if (requestCode == REQUEST_CODE) {

        if (resultCode == RESULT_OK) {

            loadWeatherData(mSharedPreferences.getString("city_code", null),
mSharedPreferences.getString("city_name_ch", null), mSharedPreferences.getString("update_time", null),
mSharedPreferences.getString("data_now", null), mSharedPreferences.getString("txt_d", null),
mSharedPreferences.getString("txt_n", null), mSharedPreferences.getString("tmp_min", null),
mSharedPreferences.getString("tmp_max", null));

        }

    }

}

//刷新各组件数据的封装

private void loadWeatherData(String city_code, String city_name, String update_time, String current_data,
String txt_d, String txt_n, String tmp_min, String tmp_max) {

    mTextView_cityName.setText(city_name);

    mTextView_updateTime.setText(update_time);

    mTextView_current_date.setText(current_data);

    if (txt_d.equals(txt_n)) {

        mTextView_weather_desp.setText(txt_d);

    } else {

        mTextView_weather_desp.setText(txt_d + "转" + txt_n);

    }

    mTextView_textView_temp1.setText(tmp_min + "°C");

    mTextView_textView_temp2.setText(tmp_max + "°C");

    mCity_current.setCity_name_ch(city_name);

    mCity_current.setCity_code(city_code);

}
```



```
}

//从服务器更新数据（CityChooseActivity中有相似方法）

private void updateWeatherFromServer() {

    String address = "https://api.heweather.com/x3/weather?cityid=" + mCity_current.getCity_code() +
"&key=" + MaoWeatherActivity.WEATHER_KEY;

    showProgressDialog();

    HttpUtil.sendHttpRequest(address, new HttpCallback() {

        @Override

        public void onFinish(final String response) {

            runOnUiThread(new Runnable() {

                @Override

                public void run() {

                    if (Utility.handleWeatherResponse(mEditor, response)) {

                        loadWeatherData(mSharedPreferences.getString("city_code", null),
mSharedPreferences.getString("city_name_ch", null), mSharedPreferences.getString("update_time", null),
mSharedPreferences.getString("data_now", null), mSharedPreferences.getString("txt_d", null),
mSharedPreferences.getString("txt_n", null), mSharedPreferences.getString("tmp_min", null),
mSharedPreferences.getString("tmp_max", null));

                        closeProgressDialog();

                    }

                }

            });

        }

    })

}
```

在本类中可以看到显示初始化各类控件，然后从本地查找，如果有城市代号则从本地获取数据，否则设置初始城市，从服务器更新数据 `loadWeatherData()`，方法即为从本地获取数据方法，`updateWeatherFromServer()`则是从服务器更新数据方法。很显然，此类还有些控件和方法似乎还没用到啊，这当然是有用的，是实现更换城市和更新天气的功能，接下来就继续实现这些功能！

4.4 切换城市和手动更新天气

在上面代码WeatherActivity类中，已经加入了城市切换按钮和天气更新按钮，并且设置了点击事件，当点击更新天气时，就会调用updateWeatherFromServer（）方法进行更新天气，此方法也是从服务器获取数据，并存储到本地，然后调用loadWeatherData（）方法进行显示！从服务器读取数据，在前面遍历数据的时候已经说过了！可以看到，点击切换城市按钮时，我们会跳转到CityChooseActivity类，在此类中先是判断本地有没有从服务器同步过城市数据，如果没有则调用queryCitiesFromServer（）方法同步数据，如果同步过，则调用queryCitiesFromLocal（）方法取出所有数据，但是，全国那么多城市，难道要一个个去找吗？当然不可能，如果是这样，只会让用户离你越来越远，所以在此设置了一个搜索框，匹配相似的城市，再调用queryCitiesFromLocal（）方法取出数据，而后面正是动态数组，并且将数据加载在ListView上，并设置点击事件，然后在queryWeatherFromServer（）中实现从服务器中获取天气数据，并结束该Activity。返回WeatherActivity类，并返回了RESULT_OK，在WeatherActivity类中onActivityResult（）方法中在此从本地获取数据显示到屏幕！

```
public class CityChooseActivity extends Activity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.activity_citychoose);

        mMaoWeatherDB = MaoWeatherDB.getInstance(this); //获取数据库处理对象
        mSharedPreferences = PreferenceManager.getDefaultSharedPreferences(this); //获取本地存储对象
        mEditor = mSharedPreferences.edit(); //获取本地存储对象

        //先检查本地是否已同步过城市数据，如果没有，则从服务器同步

        if (mMaoWeatherDB.checkDataState() == NONE_DATA) {

            queryCitiesFromServer();

        }

        mCities = queryCitiesFromLocal(""); //获取本地存储的所有的城市
    }
}
```




```
//搜索框，设置文本变化监听器

editText = (EditText) findViewById(R.id.edit_city);

editText.addTextChangedListener(new TextWatcher() {

    @Override

    public void beforeTextChanged(CharSequence s, int start, int count, int after) {

    }

    @Override

    public void onTextChanged(CharSequence s, int start, int before, int count) {

        mCities = queryCitiesFromLocal(s.toString()); //每次文本变化就去本地数据库查询匹配的
城市

        mAdapter.notifyDataSetChanged(); //通知更新

    }

    @Override

    public void afterTextChanged(Editable s) {

    }

});

mAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, cityNames); //适配
器初始化

mListView = (ListView) findViewById(R.id.list_view_cities);

mListView.setAdapter(mAdapter);

//ListView的Item点击事件

mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override

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

        mCity_selected = mCities.get(position); //根据点击的位置获取对应的City对象

        queryWeatherFromServer(); //根据点击的城市从服务器获取天气数据

    }

});
```

```

    });

}

//从服务器取出所有的城市信息

private void queryCitiesFromServer() {

    String address = " https://api.heweather.com/x3/citylist?search=allchina&key=" +
MaoWeatherActivity.WEATHER_KEY;

    showProgressDialog();

    HttpUtil.sendHttpRequest(address, new HttpCallback() {

        @Override

        public void onFinish(String response) {

            if (Utility.handleCityResponse(mMaoWeatherDB, response)) {

                runOnUiThread(new Runnable() {

                    @Override

                    public void run() {

                        closeProgressDialog();

                        mMaoWeatherDB.updateDataState();

                    }

                });

            }

        }

    })

    @Override

    public void onError(final Exception e) {

        runOnUiThread(new Runnable() {

            @Override

            public void run() {

                closeProgressDialog();

                Toast.makeText(CityChooseActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();

            }

        });

    });

```



```
    }

    });

}

//从本地数据库取出相似的城市名称

private List<City> queryCitiesFromLocal(String name) {

    List<City> cities = mMaoWeatherDB.loadCitiesByName(name);

    cityNames.clear();

    for (City city : cities) {

        cityNames.add(city.getCity_name_ch());

    }

    return cities;

}

//从服务器获取天气数据

private void queryWeatherFromServer() {

    String address = "https://api.heweather.com/x3/weather?cityid=" + mCity_selected.getCity_code() +
"&key=" + MaoWeatherActivity.WEATHER_KEY;

    showProgressDialog();

    HttpUtil.sendHttpRequest(address, new HttpCallback() {

        @Override

        public void onFinish(String response) {

            //将从服务器获取的JSON数据进行解析

            if (Utility.handleWeatherResponse(mEditor, response)) {

                //注意这里对线程的处理

                runOnUiThread(new Runnable() {

                    @Override

                    public void run() {

                        closeProgressDialog();

                        //处理完天气数据，说明已经保存到本地，我们不用再把数据封装到Intent
                        //里面返回给MaoWeatherActivity
                    }

                });

            }

        }

    });

}
```

```

        //可以在onActivityResult里面从本地存储中获取
        setResult(RESULT_OK);
        finish();
    }
});
}
}

@Override
public void onError(Exception e) {
    //注意这里对线程的处理
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            closeProgressDialog();
            Toast.makeText(CityChooseActivity.this, "数据同步失败",
Toast.LENGTH_SHORT).show();
        }
    });
}
});
}
}

```

4.5 后台自动更新天气

还是先上代码，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 {

    SharedPreferences sharedPreferences;

    SharedPreferences.Editor editor;

```

```
@Override

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

    new Thread(new Runnable() {

        @Override

        public void run() {

            updateWeather();

        }

    });

    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);

    int hour = 60 * 60 * 1000;

    long triggerTime = SystemClock.currentThreadTimeMillis() + hour;

    Intent intent_for_receiver = new Intent(this, AutoUpdateReceiver.class);

    PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

    alarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerTime, pendingIntent);

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

}

@Override

public void onCreate() {

    super.onCreate();

    sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);

    editor = sharedPreferences.edit();

}
```

```
@Override

public IBinder onBind(Intent intent) {

    return null;

}

private void updateWeather() {

    String city_code = sharedPreferences.getString("city_code", null);

    String address = "https://api.heweather.com/x3/weather?cityid=" + city_code + "&key=" +
MaoWeatherActivity.WEATHER_KEY;

    HttpUtil.sendHttpRequest(address, new HttpCallback() {

        @Override

        public void onFinish(String response) {

            Utility.handleWeatherResponse(editor, 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。从代码中看到，设置一小时更新一次天气，1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 intent_for_service = new Intent(context, AutoUpdateService.class);  
context.startService(intent_for_service);  
  
}  
}
```

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。在前面WeatherActivity类中已经激活过该服务了，接下来就是AndroidManifest文件的配置了，获取相关权限和注册服务于广播接收器。

4.6 系统测试

为了保证程序正常运行，测试是必不可少的，那么下面就将对软件进行一个基本的功能测试

测试环境

本次测试直接采用真机模拟，系统版本为 Android 4.4 KitKat，wifi 网络环境！

功能测试

功能测试就是检验代码实现的功能是否正常，用户能否正常使用这些功能，下面将进行一个简单测试，见下表：

序号	功能模块名称	功能指标	是否通过
A1	遍历全国城市数据	能够显示全国所有城市数据	通过
A2	显示天气信息	能够显示实时天气信息	通过
C3	更换城市	能够自由切换城市	通过
D4	更新天气	能够实时更新天气信息	通过

表 4-1 功能测试表

下面两张图分别展示了遍历全国城市数据和显示天气信息，由于更新天气是无法直

接展示出来的，就没有办法上图了，但是经测试，是可以进行更新天气信息的！



图 4-2 遍历城市



图 4-3 显示天气信息

5.结论

写到这论文总算到了收尾阶段，不得不说，这段时间自己收获不少，期间遇到的困难也不少。不得不说，写毕业设计文档对我来说不是一件容易的事，在开始选题的时候飘忽不定，这可能也是大家的通病吧，不知道该选择什么样的课题，在经过几番挣扎之后终于选择了这个课题，于是便开始撰写文档。其实在这个时候自己的代码并没有完成，没有金刚钻，揽不了瓷器活！前面的文档写的很快，直接就写到需求，侃侃而谈，似乎还可以再写好多，这就导致自己进了一个误区，写需求是要有侧重点，用户真正需要什么，而不是所有的需求都写出来，越华丽越好！等到写到功能实现发现自己力不从心，而此时代码并没有完成，也不敢贸然继续写下去，这个时候脑子特别乱，要代码没代码，要设计无设计，这个时候也是最艰难地阶段！没有办法，只有在网上查找资料，好在在网上看到相似的解决方案，然后艰难的完成了代码，但是进行真机测试时，发现数据总是同步失败，并且天气数据也完全不正确，这一下子把我打回原形。于是继续网上查找资料，这时才发现，自己所使用的天气接口早已过时，无奈之下，只好重新更换接口，但是不同接口使用方式是完全不同的，这意味着，我必须推到前面的代码，重新编写，这是一个漫长而且痛苦的过程，在重构还是重写犹豫了很久，最后决定进行重构，通过对 API 接口文档的阅读，慢慢懂得了该接口的使用，原本是省>地级市>县一级一级筛选，现在变成了关键字搜索城市进行选择，好在最后完成了代码，功能上没有太大区别，有了代码之后，编写文档的过程变得容易很多！

通过这次完成毕业设计，也大致了解了软件开发的周期，也对程序设计有了进一步了解，以前总是一个包写到底，想到哪里写哪里，从来不会从全局考虑，导致写到后面无从下手。通过这次做毕业设计，懂得了做一款软件，首先要进行需求分析，然后进行技术分析，否则会让代码写的十分盲目，准备工作一定要做充分，比如我这次没有注意到，接口已经更换，导致必须进行重构，这是大忌。老话说不打无准备之仗，就是这个意思。

软件写完并非万事大吉，还得进行程序测试，因为代码不报错，不代表可以完美运行，功能是否完全正常，甚至程序是否可以正常运行，这都是需要进行一系列的测试才能得出结论的，如果将一款并不完善的软件推向市场，是会流失用户的。

完成这次毕业设计之前，从来没有这么完整的走完一个软件开发周期，这让我学到了很多，纠正了我以前的一些错误思想。也让我懂得了如何使用第三方 api 接口，对自己进行了一次拓展，也让我受益匪浅。

致谢

在本次毕业设计过程中,得到了老师和同学的指导和支持,再次感谢老师和同学的帮助,指导老师悉心指导,在选题和设计上都给了很好的建议,及时答疑解惑,也让自己少走了一些弯路,让此次毕业设计能够顺利完成,再次表示特别感谢!

参考文献

- [1]唐磊. 浅谈 SQLite 数据库技术在 Android 平台的应用[J]. 电子世界, 2014(9):12-13.
- [2] 宋俊. 安卓手机 APP 开发之 HTTP 通信[J]. 通讯世界, 2016(18).
- [3] 李洋, 殷云鹏, 赵勇. 基于 Android 的网络数据存储与访问[J]. 中国科技信息, 2013(8):92-92.
- [4] 谭翔纬. 利用 JSON 实现 Android 客户端与 Web 服务器间的数据交互[J]. 福建电脑, 2013, 29(2):166-167.
- [5] 郭霖. 第一行代码 Android[M]. 人民邮电出版社, 2014.
- [6] 王康, 李纪欣. Android 平台休眠唤醒机制研究[J]. 计算机工程, 2011(S1):320-323.
- [7] 林培杰, 朱安南, 程树英. Android 数据库 SQLite 性能优化[J]. 计算机系统应用, 2014, 23(4):193-196.
- [8] 李侠, 沈峰. Android 存储机制的应用研究[J]. 电脑知识与技术:学术交流, 2013(24):5535-5538.