**特色**

1、使用OkHttp获取网络数据

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

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

dependencies {

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

}

2、使用GSON解析JSON格式数据

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

dependencies {

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

}

GSON将JSON数据的对象和一个实体类相映射。例如一段JSON格式数据如下所示：{“name”:”Tom”,”age”:20}，那我们只需要定义一个Person类，并加入name和age，并调用fromJson(jsonData,Person.class)即可自动获得一个Person对象。

3、使用LitePal与本地数据库进行交互

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

LitePal采用了对象关系映射(ORM)模式，并将常用数据库功能进行了封装，使得开发者可以简单快捷的完成数据库操作。

4、使用SharedPreferences进行本地缓存

我们需要在用户每次打开软件后加载用户之前选择的地点的天气信息，但是我们认为同一地点的天气状况在一个较长的时间之中是不会发生变化的，所以我们不需要每次都要去服务器获取数据，因此我们在每次获取过服务器的信息后使用SharedPreferences将数据本地缓存。因此当用户再次打开后可以直接通过解析本地缓存的方式来获取数据，从而减小了访问数据库的次数。

5、使用Alarm机制定时后台更新

我们需要一个后台服务定期对天气信息进行更新，这里我们使用了Alarm机制，没有采用Timer的原因是Timer并不能唤醒CPU，因此它有可能会因为长时间不活跃被系统回收，而Alarm是具有唤醒CPU的能力的。使用Alarm每隔8小时去服务器获取一个数据，并更新本地缓存，这样在下次启动时就可以获得较新的数据。