移动互联网技术及应用

大作业报告

姓名	班级	学号
罗天佑	**2018211307**	**2018211349**

2021.6

目录

1. 相关技术

1.1. 网络

1.1.1. 网络数据源使用 Retrofit 库访问彩云 API 提供的 Webservice 接口来实现。Retrofit 通过封装网络请求和数据解析，极大地提升了开发效率。并且支持自定义数据解析

1.1.2. 在封装所有网络请求的 API 时，我使用了协程技术来简化 Retrofit 回调的写法。

1.2. 数据存储

1.2.1. 本地数据源使用 SharedPreferences 持久化技术来实现，使用键值对的方式来存储数据。相较于数据库保存数据，SharedPreferences 操作更加简单，本地需要保存的数据也比较简单。因此 SharedPreferences 更加合适。

1.3. UI

1.3.1. 创建 ViewModel 实现 ui 层与 service 层的解耦合

1.3.2. 使用滚动控件 RecyclerView，使天气展示界面可以轻松高效地展示大量数据。我提供数据并定义每个列表项的外观，而 RecyclerView 库会根据需要动态创建元素

1.3.3. 使用 Fragment，使我的一个单独的 activity 可以合成一个多区域的 UI，并且可以在多个 activity 中再使用。

1.4. 全局获取 Context 对象

1.4.1. 由于本项目需要在 Activity 以外的很多地方使用 Context 对象。因此我使用 Application 类，在应用程序启动的时候，系统就会自动将其进行初始化，以便于我们管理程序内如全局 Context 一样的一些全局的状态信息。

```
class SunnyWeatherApplication : Application() {

    companion object {

        const val TOKEN = "Barmss6X0pJtLgf0" // 彩云天气API的令牌

        @SuppressLint( ...value: "StaticFieldLeak")
        lateinit var context: Context // 定义了一个context变量
    }

    override fun onCreate() {
        super.onCreate()
        // applicationContext是getApplicationContext()方法得到的返回值
        context = applicationContext
    }

}
```

这样就可以以静态变量的形式获取 Context 对象了。

同时项目中还需要在在 AndroidManifest.xml 文件的<application>标签下进行指定自定义的 Application 类。

```
<application
    android:name=".SunnyWeatherApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
```

2. 系统功能需求

2.1. 需求分析

本项目实现了一个简单的天气信息查询的 APP。作为查询信息类的 APP 为了可以是用户能够及时准确的获取一个城市的天气信息。APP 至少应该具备准确查询城市级天气信息、可以查询多城市天气、可以刷新实时天气这三项基本功能。

由于获取城市天气信息的功能过于复杂，且与本课程学习重点无关。因此，我使用了彩云天气的开放 API 获取城市的天气。该 API 可以获取全球 100 多个国家的城市数据，以及每个城市的实时天气预报信息。经过简单的申请注册即可获取令牌使用。

2.2. 功能描述

因此本项目具有：

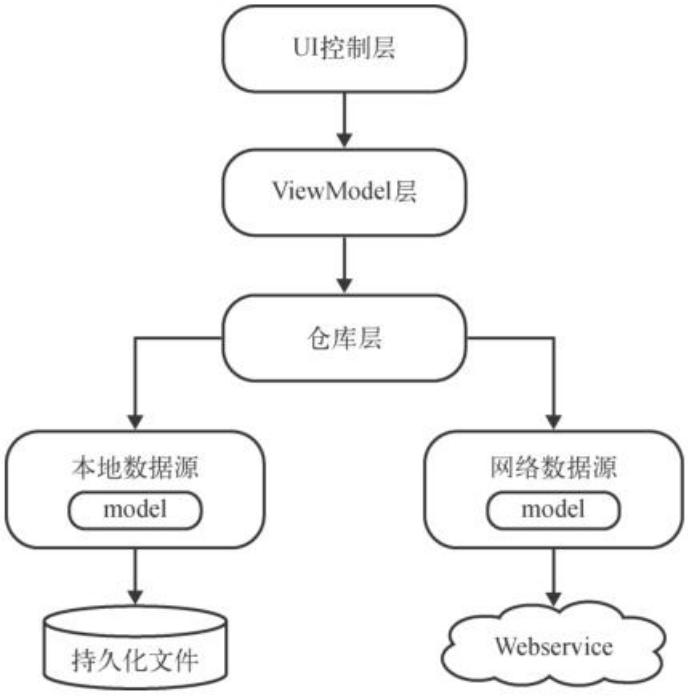
1. 可以搜索全球大多数国家的各个城市数据；
2. 可以查看全球绝大多数城市的天气信息；
3. 可以自由地切换城市，查看其他城市的天气；
4. 可以手动刷新实时的天气。

这四大主要功能，足以满足用户日常查询天气的需求。

3. 系统设计与实现

3.1. 总体设计

在本项目中，我使用了 MVVM 架构模式。MVVM 架构将程序结构主要分为 Model、View、ViewModel 三个模块。其中 Model 是数据模型部分；View 是界面展示部分；而 ViewModel 实现了业务逻辑和界面展示的解耦合。



MVVM 项目架构示意图

UI 控制层包含了平时写的 Activity、Fragment、布局文件等与界面相关的展示代码。

ViewModel 层用于对 UI 元素相关的数据进行持久化，同时 ViewModel 层还提供了接口给 UI 控制层调用以及和 model 层进行通信。

Model 层用于判断调用方请求的数据应该是从本地数据源中获取还是从网络数据源中获取，并将获取到的数据返回给调用方。实现了 APP 所必须的逻辑处理功能。

3.2. 系统组成

按照 MVVM 架构的设计，我在项目中将设计业务逻辑处理（model 层）的代码放在 service 包，并且将界面展示部分（view 层）代码放在 ui 包中。

3.3. 模块设计

3.3.1. service 模块

在 logic 包下新建一个 Repository 单例类，作为仓库层的统一封装入口。使用 liveData()将异步获取的数据以响应式编程的方式通知给上一层。同时我将 liveData()函数的线程参数类型指定成了 Dispatchers.IO 使这段网络请求代码运行在子线程。

```
object Repository {

    // 本代码段在子线程上运行
    fun searchPlaces(query: String) = fire(Dispatchers.IO) { // fire : LiveData<Result<T>>
        val placeResponse = SunnyWeatherNetwork.searchPlaces(query)
        if (placeResponse.status == "ok") {
            val places = placeResponse.places
            Result.success(places)
        } else {
            Result.failure(RuntimeException("response status is ${placeResponse.status}"))
        }
    }
```

在 logic/model 包下新建一个 Sky.kt 文件，定义了一个 Sky 类作为数据模型表示该天气情况所对应的文字、图标和背景。然后使用 mapOf()函数来定义每种天气代码所应该对应的文字、

图标和背景。

```
private val sky = mapOf(
    "CLEAR_DAY" to Sky( info: "晴", R.drawable.ic_clear_day, R.drawable.bg_clear_day),
    "CLEAR_NIGHT" to Sky( info: "晴", R.drawable.ic_clear_night, R.drawable.bg_clear_night),
```

在 logic/model 通过 PlaceResponse.kt 文件定义数据模型实现对城市位置信息的映射。

在 logic/model 包下新建一个 RealtimeResponse.kt 文件定义数据模型实现对当前天气信息的映射。

在 logic/model 包下新建一个 DailyResponse.kt 文件定义数据模型实现对预测天气信息的映射。

在 logic/model 包下定义一个 Weather 类，将 Realtime 和 Daily 对象封装起来。

```
data class Weather(val realtime: RealtimeResponse.Realtime, val daily: DailyResponse.Daily)
```

在 logic/network 包下新建一个 ServiceCreator 单例类，创建了一个 Retrofit 构建器。

在 logic/network 包下新建 PlaceService 接口，定义一个用于访问彩云天气城市搜索 API 的 Retrofit 接口。

```
interface PlaceService {

    @GET( value: "v2/place?token=${SunnyWeatherApplication.TOKEN}&lang=zh_CN")
    fun searchPlaces(@Query( value: "query") query: String): Call<PlaceResponse>

}
```

该函数的返回值为 Call<PlaceResponse>，这样 Retrofit 就会将服务器返回的 JSON 数据自动解析成 PlaceResponse 对象了。

在 logic/network 包下新建 WeatherService 接口定义一个用于访问天气信息 API

的 Retrofit 接口。

在 logic/network 包下新建一个 SunnyWeatherNetwork 单例类，定义一个统一的网络数据源访问入口，对所有网络请求的 API 进行封装。

```
private val weatherService = ServiceCreator.create(WeatherService::class.java)

suspend fun getDailyWeather(lng: String, lat: String) = weatherService.getDailyWeather(lng, lat).await()

suspend fun getRealtimeWeather(lng: String, lat: String) = weatherService.getRealtimeWeather(lng, lat).await()

private val placeService = ServiceCreator.create(PlaceService::class.java)

suspend fun searchPlaces(query: String) = placeService.searchPlaces(query).await()
```

并自定义的 await()函数调用 suspendCoroutine 函数简化回调过程。

```
private suspend fun <T> Call<T>.await(): T {
    return suspendCoroutine { continuation -> // 在一个普通的线程中执行下列代码
        enqueue(object : Callback<T> {
            override fun onResponse(call: Call<T>, response: Response<T>) {
                val body = response.body()
                if (body != null) continuation.resume(body)
                else continuation.resumeWithException(RuntimeException("response body is null"))
            }

            override fun onFailure(call: Call<T>, t: Throwable) {
                continuation.resumeWithException(t)
            }
        })
    }
}
```

在此当外部调用 SunnyWeatherNetwork 所提供的函数时，Retrofit 就会立即发起网络请求，同时当前的协程也会被阻塞住。直到服务器响应我们的请求之后，await()函数会将解析出来的数据模型对象取出并返回，同时恢复当前协程的执行，searchPlaces()函数在得到 await()函数的返回值后会将该数据再返回到上一层。

在 logic/dao 包下新建一个 PlaceDao 单例类封装了几个必要的存储和读取数据的接口，将 Place 对象存储到 SharedPreferences 文件中，并编写一些涉及 SharedPreferences 的操作。

```
fun savePlace(place: Place) {
    sharedPreferences().edit {  this: SharedPreferences.Editor
        putString("place", Gson().toJson(place)) // 通过GSON将Place对象转成一个JSON字符串,用字符串存储的方式来保存数据
    }
}
```

3.3.2. ui 模块

实现搜索城市界面：

在 ui/place 包下新建一个 PlaceViewModel，定义搜索城市界面的 ViewModel 层。
在 ui/place 包下新建一个 PlaceAdapter 类为 RecyclerView 准备适配器。
在 ui/place 包下新建一个 PlaceFragment 对 Fragment 进行实现。

实现天气展示界面：
在 ui/weather 包下新建一个 WeatherViewModel，定义天气展示的 ViewModel 层。
在 ui/weather 包下新建一个 WeatherActivity，显示天气信息的 Activity

4. 系统可能的扩展

由于时间紧迫，本项目只是简单的实现了最基础的一些功能。目前可以想到的扩展如下：

4.1. 提供更加完整的天气信息

4.2. 允许选择多个城市，观察多个城市的天气信息

4.3. 添加用户个性化选择，选择查看的天气信息

4.4. 对深色主题进行适配

5. 总结体会

通过本学期的学习以及本次大作业的代码编写，我的 Android 开发水平和对于移动互联网相关知识的了解得到了进一步的提高。能够独立编写出一些有意思的 APP 也使我更有动力继续学习安卓开发的相关知识。在编写代码的过程中，新的问题层出不断，在我分析问题解决问题的过程中，我对于课上所教授的知识有了更加深刻的理解。同时，我也了解到了一些课上未曾讲过的高级语法，简化了我解决网络等方面遇到的问题。