目录

[渠道号 2](#_Toc23516)

[IMEI 国际移动设备识别码 （International Mobile Equipment Identity）。 2](#_Toc13309)

[外广？外部广告 2](#_Toc10928)

[@SuppressLint("NewApi", "WrongConstant") 2](#_Toc31297)

[安卓手机的各种栏 2](#_Toc7618)

[Android application类 3](#_Toc13589)

[decorView 3](#_Toc73)

[DrawerLayout 抽屉布局 3](#_Toc26388)

[Dialog对话 3](#_Toc9650)

[NestedScrollView 3](#_Toc27718)

[Adapter的理解 4](#_Toc16108)

[ViewHolder 4](#_Toc2760)

[查看设备的qaid 4](#_Toc30460)

[Base基类（MVP）作用： 4](#_Toc12661)

[冷启动、热启动的区别 4](#_Toc31097)

[RemoteViews的应用 5](#_Toc7438)

[tools:text="I am a title" 5](#_Toc22941)

[PendingIntent 5](#_Toc8055)

[ArrayList 6](#_Toc18179)

[Lottie 6](#_Toc25088)

[TextView与Button的区别 6](#_Toc9560)

[AbstractActivity 6](#_Toc724)

[Context 7](#_Toc1666)

[View 9](#_Toc12036)

[UI管理系统的层级关系 10](#_Toc17161)

[Adapter、Holder之类的逻辑与架构 12](#_Toc9675)

[CityDisplayModel 13](#_Toc14068)

[Holder中的Init和bindData 13](#_Toc22581)

[Adapter中的setData和onCreate和onBind 14](#_Toc7574)

[MainViewModel 14](#_Toc17079)

[setUserVisibleHint 14](#_Toc24956)

[24小时天气数据（或者所有的天气数据） 14](#_Toc20709)

[RecyclerView中的Fragment的生命周期 15](#_Toc31983)

[纯净包和外广包的区别？ 15](#_Toc12720)

[cityWeatherFragment和WeatherFragment分别是做什么的？ 16](#_Toc12739)

[RainMapOftenWeatherHolder.kt 最近的降雨带模块 17](#_Toc15285)

[空气质量页有两个fragment在主包下 17](#_Toc4050)

[对项目结构的理解 17](#_Toc19534)

[测试包和正式包和渠道包在哪里打 18](#_Toc77)

# 渠道号

1今日头条

62vivo

65应用宝

# IMEI 国际移动设备识别码 （International Mobile Equipment Identity）。

由15位数字组成，全球唯一ID，用来区别每一部移动通信设备。与手机是绑定关系。

第一部分 TAC，Type Allocation Code，类型分配码。86为中国。

其余的部分没啥有用信息。

典型IMEI码：868540050954128

# 外广？外部广告

ActivityChannel 活动渠道/归因渠道 用户的正确的来源渠道

# @SuppressLint("NewApi", "WrongConstant")

可以使用****@SuppressLint****标注忽略指定的警告。

我们有时会使用比我们在AndroidManifest中设置的android:minSdkVersion版本更高的方法，此时编译器会提示警告，解决方法是在方法上加上@SuppressLint("NewApi"）或者@TargetApi()。  
@SuppressLint("NewApi"）屏蔽一切新api中才能使用的方法报的android lint错误

# 安卓手机的各种栏

  
Ab接口

1. b对照组，也就是灰度，为了将用户分为2组来判断新上线的功能能不能带来正向反馈。

使用：调用接口后判断是哪个组的之后做对应的需求就可以

# **Android application类**

# decorView

getWindow().getDecorView()的方法可以获取到decorView，decorView是什么呢？

decorView是window中的最顶层view，可以从window中获取到decorView，然后decorView有个getWindowVisibleDisplayFrame方法可以获取到程序显示的区域，包括标题栏，但不包括状态栏。

# DrawerLayout 抽屉布局

# Dialog对话

# NestedScrollView

# Adapter的理解

Adapter是将RecyclerView的元素加载进去，具体用哪个RecyclerView在外部引用这个Adapter就可以了

# ViewHolder

ViewHolder是用来装item的地方，是一个内部类，里面也有属性，需要和数据源（item类）绑定

在点击事件里面holder.itemView.setOnClickListener{

}

ItemView是所有的视图，所以会（循环？）遍历所有的子view，在里面可以设置条件if (holder.itemText.text == next){

return@setOnClickListener

}

来控制点击时间

# 查看设备的qaid

打开app后在日志内搜索toolnetwork 中的qaid

# Base基类（MVP）作用：

1. 减少代码重复

# 冷启动、热启动的区别

冷启动：

在启动应用时，系统中没有该应用的进程，这时系统会创建一个新的进程分配给该应用；

热启动：

在启动应用时，系统中已有该应用的进程(例：按back键、home键，应用虽然会退出，但是该应用的进程还是保留在后台)；

# RemoteViews的应用

桌面小部件则是通过AppWidgetProvider来实现的，AppWidget本质是一个广播.

通知栏和桌面小部件的开发过程中都会用到RemoteView,它们在更新界面时无法像在Activity里面那样直接更新View,这是因为两者的界面都运行在其他线程中，确切的说是系统的SystemServer进程.为了跨进程更新界面，RemoteViews提供一系列set方法，并且这些方法只是View全部方法的子集，另外RemoteVIew支持的View类型也是有限的。

# tools:text="I am a title"

tools[命名空间](https://so.csdn.net/so/search?q=%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/CHS007chs/article/details/_blank)比较特别，该命名空间可以覆盖组件的任何属性，以便在ide预览中进行不同的展示，。

# PendingIntent

pendingIntent必须要满足一定条件下，才会触发放在里面的intent操作。可以在程序外部进行执行。即使是程序已经退出了。

在Android中，我们常常使用PendingIntent来表达一种“留待日后处理”的意思。从这个角度来说，PendingIntent可以被理解为一种特殊的异步处理机制。不过，单就命名而言，PendingIntent其实具有一定误导性，因为它既不继承于Intent，也不包含Intent，它的核心可以粗略地汇总成四个字——“异步激发”。  
很明显，这种异步激发常常是要跨进程执行的。比如说A进程作为发起端，它可以从系统“获取”一个PendingIntent，然后A进程可以将PendingIntent对象通过binder机制“传递”给B进程，再由B进程在未来某个合适时机，“回调”PendingIntent对象的send()动作，完成激发。

PendingIntent 是 Android 提供的一种用于外部程序调起自身程序的能力，生命周期不与主程序相关。外部程序通过 PendingIntent 只能调用起三种组件：

* Activity
* Service
* Broadcast

PendingIntent 的使用场景有三个：

* 使用 AlarmManager 设定闹钟
* 在系统状态栏显示 Notification
* 在桌面显示 Widget

# ArrayList

ArrayList.removeAll(Collection<?> c) 去除当前列表中所有与传入Collection有交集的元素

ArrayList.clear() 清空所有元素

初始化：

val list15Days: ArrayList<Forecast15DayBean> = ArrayList()

# Lottie

Lottie 是 Airbnb 开发的一款能够为原生应用添加动画效果的开源工具。可以直接通过json文件来调用不同的动画效果。

# TextView与Button的区别

Button继承于TextView，一般情况下，TextView用来承载静态文本，Button用来响应点击事件。但是TextView 继承自View，点击事件是从View就有的特性，所以TextView也能点击。那么TextView与Button有什么不同呢？

1.TextView中文字默认是从左上方开始，Button中文字默认是居中显示

2.TextView默认背景透明，Button默认背景灰色的

3.TextView点击无效果，Button默认状态下点击变黄色（它有一个默认的背景选择器）

4.TextView默认不拦截点击事件，Button会拦截点击事件(比如说我们一个ListView，Item的布局中有一个TextView和Button，我们在这个TextView上点击会触发ListView的OnItemClickListener方法，在Button上点击并不会触发ListView的OnItemClickListener方法，因为Button把这个点击事件拦截了)。

# AbstractActivity

Activity继承自AbstractActivity<ActivityClockRemindBinding>(){

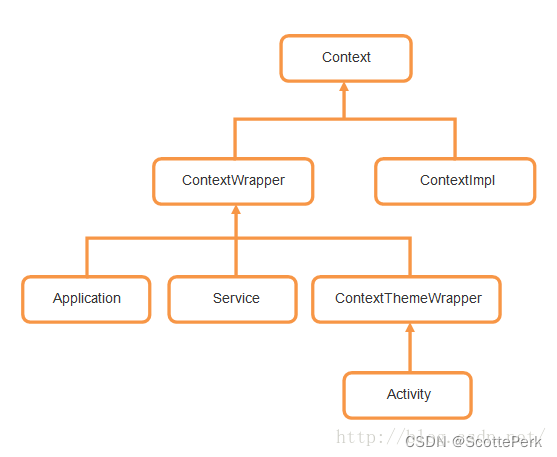
重写getBinding用来绑定视图

重写initView用来初始化View。如：adapter，layoutManager，onSingleClick等

重写initData用来做数据操作。如：对接Model，向列表添加数据等

}

# Context



Context到底是什么

Context的中文翻译为：语境; 上下文; 背景; 环境，在开发中我们经常说称之为“上下文”，Android程序员把“场景”抽象为Context类，他们认为用户和操作系统的每一次交互都是一个场景，比如打电话、发短信，这些都是一个有界面的场景，还有一些没有界面的场景，比如后台运行的服务（Service）。

Android程序不像Java程序一样，随便创建一个类，写个main()方法就能运行，Android应用模型是基于组件的应用[设计模式](https://so.csdn.net/so/search?q=%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/xielongfeiabcd/article/details/_blank)，组件的运行要有一个完整的Android工程环境，在这个环境下，Activity、Service等系统组件才能够正常工作，而这些组件并不能采用普通的Java对象创建方式，new一下就能创建实例了，而是要有它们各自的上下文环境，也就是我们这里讨论的Context。可以这样讲，Context是维持Android程序中各组件能够正常工作的一个核心功能类。

源码中的注释是这么来解释Context的：Context提供了关于应用环境全局信息的接口。它是一个抽象类，它的执行被Android系统所提供。它允许获取以应用为特征的资源和类型，是一个统领一些资源（应用程序环境变量等）的上下文。

Activity，Application，Service虽都继承自ContextWrapper（Activity继承自ContextWrapper的子类ContextThemeWrapper），但它们初始化的过程中都会创建ContextImpl对象，由ContextImpl实现Context中的方法。

Broadcast Receiver，Content Provider并不是Context的子类，他们所持有的Context都是其他地方传过去的，所以并不计入Context总数。

绝大多数场景下，Activity、Service和Application这三种类型的Context都是可以通用的。不过有几种场景比较特殊，比如启动Activity，还有弹出Dialog。出于安全原因的考虑，Android是不允许Activity或Dialog凭空出现的，一个Activity的启动必须要建立在另一个Activity的基础之上，也就是以此形成的返回栈。而Dialog则必须在一个Activity上面弹出（除非是System Alert类型的Dialog）



Activity继承自ContextThemeWrapper，而Application和Service继承自ContextWrapper，很显然ContextThemeWrapper在ContextWrapper的基础上又做了一些操作使得Activity变得更强大

一句话总结：凡是跟UI相关的，都应该使用Activity做为Context来处理；其他的一些操作，Service,Activity,Application等实例都可以，当然了，注意Context引用的持有，防止内存泄漏。

getApplication()和getApplicationContext()的区别

Application本身就是一个Context，所以这里获取getApplicationContext()得到的结果就是Application本身的实例。实际上这两个方法在作用域上有比较大的区别。getApplication()方法的语义性非常强，一看就知道是用来获取Application实例的，但是这个方法只有在Activity和Service中才能调用的到。 那么也许在绝大多数情况下我们都是在Activity或者Service中使用Application的，但是如果在一些其它的场景，比如BroadcastReceiver中也想获得Application的实例，这时就可以借助getApplicationContext()方法了。

如何避免Context造成内存泄漏

当Context销毁的时候，却因为被引用导致销毁失败，而Application的Context对象可以理解为随着进程存在的，所以我们总结出使用Context的正确姿势：

1：当Application的Context能搞定的情况下，并且生命周期长的对象，优先使用Application的Context。

2：不要让生命周期长于Activity的对象持有到Activity的引用。

3：尽量不要在Activity中使用非静态内部类，因为非静态内部类会隐式持有外部类实例的引用，如果使用静态内部类，将外部实例引用作为弱引用持有。

getContext(),getBaseContxet(),getApplicationContext的区别

Activity,Service和Application都有getBaseContxet(),getApplicationContext()这两个方法。但没有getContext方法。在Fragment中，才有getContext方法。而Fragment的getContext方法获取的对象是他的寄主对象，也就是Activity。

# View

**View 的位置参数**

left：View 父容器左上角横坐标

top：View 父容器左上角纵坐标

right：View 父容器右下角横坐标

bottom：View 父容器右下角纵坐标

x 和 y 代表的是 View 左上角的坐标。x 轴 和 y 轴的正方向分别为右和下。

translationX 和 translationY 是 View 左上角相对于父容器的偏移量。

x=left+translationX

**MotionEvent**

在手指接触屏幕后所产生的一系列事件中，典型的时间类型有如下三种：

ACTION\_DOWN：手指刚接触屏幕的一瞬间。

ACTION\_MOVE：手指在屏幕上移动。

ACTION\_UP：手指从屏幕上松开的一瞬间。

|  |  |
| --- | --- |
| getX() / getY() | 返回相对于当前 View 的左上角 x 和 y 坐标。 |
| getRawX() / getRawY() | 返回相对于手机屏幕左上角的 x 和 y 坐标。 |

**TouchSlop**

TouchSlop 是系统所能识别出的被认为是滑动的最小距离。

ViewConfiguration.get(context).getScaledTouchSlop()

**VelocityTracker**

VelocityTracker 用于做手指在滑动过程中的速度追踪，包括水平和竖直方向的速度。

使用：

首先，在 View 的 onTouchEvent 方法中追踪当前单击事件的速度

val velocityTracker = VelocityTracker.obtain()

velocityTracker.addMovement(event)

接着，当我们想知道当前的滑动速度时，这个时候可以采用如下方法来获得当前速度：

// 计算滑动速度，计算的时间间隔为 1000ms

// 也即，计算 1000ms 内手指所滑动的像素数

velocityTracker.computeCurrentVelocity(1000)

val xVelocity = velocityTracker.getXVelocity()

val yVelocity = velocityTracker.getYVelocity()

当手指从右往左滑动时，水平速度即为负值

在不需要使用它时，需要调用clear()方法来重置并回收内存

velocityTracker.clear()

velocityTracker.recycle()

**GestureDetector**

用于辅助检测用户的单击、滑动、长按、双击等行为。一般情况下我们需要处理用户手指的单击、长按和滑动操作，可以直接在 onClick()、onLongClick() 和 onTouchEvent() 中处理即可，如果还需要 View 处理用户的双击操作，则需要借助 GestureDetector。

**scrollTo和scrollBy**

为了实现 View 的滑动，View 提供了专门的方法来实现这个功能，那就是 scrollTo() 和 scrollBy()，

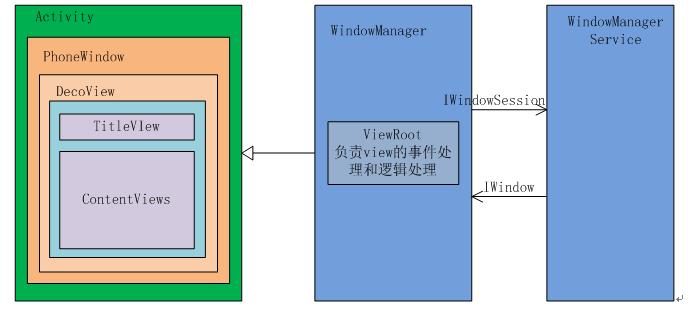
mScrollX：View 左边缘-内容左边缘的在水平方向的距离

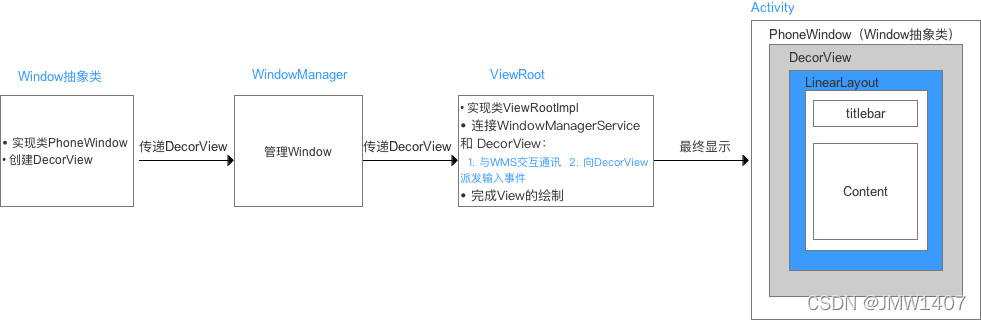
mScrollY：View 上边缘和内容上边缘的在水平方向的距离

**Scroller**

Scroller 是一个对象，用于实现 View 的弹性滑动（一定的时间间隔内，不是瞬时的）。

# UI管理系统的层级关系





PhoneWindow：Android系统中最基本的窗口系统，继承自Windows类，负责管理界面显示以及事件响应。它是Activity与View系统交互的接口

DecorView：PhoneWindow中的起始节点View，继承于View类，作为整个视图容器来使用。用于设置窗口属性。它本质上是一个FrameLayout

ViewRoot：Activtiy启动时创建，负责管理、布局、渲染窗口UI等等

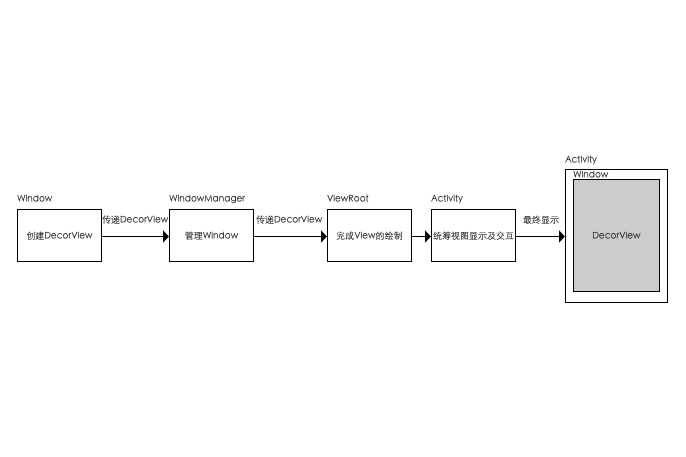
**ViewRoot与DecorView**

ViewRoot是连接WIndowManager和DecorVIew的纽带。View的三大流程(measure layout draw)都是通过ViewRoot完成的。

在ActivityThread中，Activity创建后，DecorView会被添加到Window中，同时创建ViewRootImpl，然后将DecorView与ViewRootImpl建立关联(通过ViewRoot的setView方法)。

**ViewRoot、DecorView、Window和Activity的关系**





****“activity，window，View 三者之间的关系是什么？”****

1、window 是 activity 里的一个实例变量，本质是一个接口，唯一的实现类是 PhoneWindow。

2、activity 的 setContentView 方法实际上是就是交给 phonewindow 去做的。window 和 View的关系可以类比为显示器和显示的内容。

3、每个 activity 都有一个“显示器” window，“显示的内容”就是DecorView。这个“显示器”定义了一些方法来决定如何显示内容。比如 setTitleColor setTitle是设置导航栏的颜色和 title , setAllowReturnTransitionOverlap 设置进/出场动画等等。

****所以 window 是 activity 的一个成员变量，window 和 View 是“显示器”和“显示内容”的关系。****

# Adapter、Holder之类的逻辑与架构

AirQualityAdapter是整个页面的adapter，在里面读取不同模块的holder，并在oncreatViewHolder里加载模块的布局，在onBindViewHolder里绑定数据

每个模块会有一个holder对应，holder内只管理模块中的东西，不需要绑定布局，布局在外部的也就是上面的adapter中绑定就行，holder中做绑定控件id操作，编写加载数据的方法

adapter的任务是将外布局和模块布局适配，所以每一个模块布局对应一个holder，然后外布局有一个adapter来将模块的holder加载进去（其实就是外布局是一个RecyclerView然后通过holder将布局加入进去）

adapter中的oncreateViewHolder的参数中需要传入父布局的View和种类ViewType，这样就可以使用父布局.findViewById进行交互，并且可以通过ViewType判断当前需要创建哪个Holder

adapter中的onBindViewHolder需要传入holder和position，这样可以通过判断当前的holder来绑定对应的数据，并且通过position来获取holder中的子view的位置信息控制子view

weather-xmiles-weather-fragment-cityWeatherFragment是用来加载天气界面的

在写fragment的时候继承abstractFragment就可以，layoutbaseFragmenty里面是有对dialog（弹窗）的处理，而abstract里面有binding

添加模块：

1、在adapter的setData里面在mTypeList中add来添加模块的标签

2、在adapter的onCreateViewHolder里面通过标签来判断传入的holder并创建holder的实例  
3、在onBindViewHolder中通过判断传入的holder类型来设置数据

4、重写getItemViewType方法，返回mTypeList[position]

在holder（baseHolder）里面做了：（以RainMapOftenWeatherHolder为例）

1. 加载布局（
2. 绑定控件和数据源（先用binding绑定了布局） （bindData方法）

private var mBinding = HolderRainMapOftenweatherBinding.bind(itemView)

Tips：

1、每次设置或者更新数据之后都要调用一次notifyDataSetChanged()来刷新数据

# CityDisplayModel

里面包含最新的城市的cityCode和cityName

View参数可以使用inflate加载布局来作为View传入

# Holder中的Init和bindData

init会比bindData先加载所以在init内只能做绑定和事件操作，不能做数据操作，数据操作放到bindData里面去

Init里面可以绑定Manager、adapter、RV.addItemDecoration

# Adapter中的setData和onCreate和onBind

onCreate是用来创建holder的，可以执行绑定布局，相当于用绑定的布局创建新的holder

onBind是用来绑定布局内的东西的，在这里将holder中的控件和adapter中的数据一一对应。onBind中可以调用setData。

SetData是用来从外部传入数据的（让外部调用之后数据作为参数传入），所有接触到真实数据的代码都在这。

Tips：Holder中的真实数据仅仅只用来处理Holder中的控件的数据。哪个Holder处理什么数据要分清楚。

# MainViewModel

加载底部Tab用的

# setUserVisibleHint

每次到fragment显示的时候会调用，用来设置fragment自动刷新

# 24小时天气数据（或者所有的天气数据）

都可以从appCityWeatherViewModel里获取，具体的可以使用以下方法从回调中获取数据：

appCityWeatherViewModel.loadWeatherPageData(mCityCode)

appCityWeatherViewModel.weatherPageDataBean.observe(this){

if (it != null) {

forecast24HourWeathers = it.forecast24HourWeather.forecast24HourWeathers

}

}

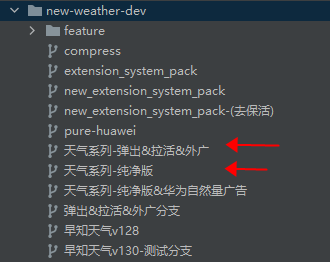
其中的this是fragment

# RecyclerView中的Fragment的生命周期

关键几点：

1. setUserVisibleHint 方法是最先走的. 首次进入的时候同一个Fragment的setUserVisibleHint 要走两次 一次false, 一次.true
2. 被预加载的Fragment的生命周期 除了setUserVisibleHint true没走之外 其他的生命周期也走了.
3. 预加载的Fragment 到显示的时候 其实只走了 setUserVisibleHint true.
4. Fragment 销毁的时候 只走到了onDestroyView方法 并没有走onDestroy onDetach方法. 这点对于 执行一些回收操作非常有必要了解.

# 纯净包和外广包的区别？

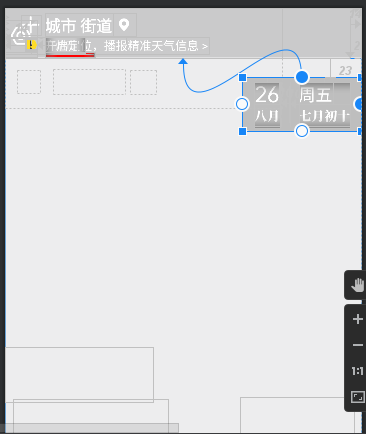


这个分支下的就是纯净包和外广包

纯净包是什么广告都没有的，外广包会弹出外广

# cityWeatherFragment和WeatherFragment分别是做什么的？

WeatherFragment天气页面：天气最底层的页面，包含顶部的标题栏、旁边的悬浮广告、右上角的日历（可隐藏）



cityWeatherFragment天气子页面：包含刷新的操作，定时刷新、测试改变天气（右上角的改变天气按钮），设置天气顶部背景（动画）及高度（switchWeatherBackground）、组件引导



# RainMapOftenWeatherHolder.kt 最近的降雨带模块

时时的有把预警信息包括在里面的

# 空气质量页有两个fragment在主包下

AirQualityFragment（其他）和EnjoyWeatherAirQualityFragment（享看和夏雨（新））

Id是不同的

# 对项目结构的理解

把fragment放到主包里，然后把模块holder放到变体里，通过router来加载模块，router会优先加载变体内同名的文件之后才加载主包内同名的文件。

如果主模块中已经有一个同名的模块则在变体中的模块可以重名，这样就会覆盖主包中的模块了。

只有继承baseHolder的holder才能放在子包用router去加载，因为在adapter的bind里面如果不用baseHolder的加载方式就需要引入本类，那么就需要在同一个包下，就没法放在子包里了

# 测试包和正式包和渠道包在哪里打

测试包和正式包在jenkins中打debug和release

渠道包才在商业化后台的渠道打包配置中打