目录

[开发流程： 3](#_Toc10309)

[DrawerLayout 抽屉布局 3](#_Toc17506)

[Tab 3](#_Toc21024)

[decorView 4](#_Toc19859)

[Dialog对话 4](#_Toc19552)

[NestedScrollView 4](#_Toc7928)

[kotlin 实现静态的方式 4](#_Toc4838)

[ViewHolder 5](#_Toc10733)

[JSON 5](#_Toc10897)

[对应JSON对象字符串，JSON数组字符串 5](#_Toc15590)

[第二个参数可以传入不同类型的值 6](#_Toc22744)

[FastJSON 6](#_Toc7460)

[如何获取现在已经点击的view？ 6](#_Toc5386)

[recyclerView的理解 6](#_Toc30567)

[通过位置信息和设置来绑定控件和数据 7](#_Toc26837)

[1、Xml添加RecyclerView 7](#_Toc9962)

[1、 初始化数据fruitList 7](#_Toc25027)

[1、 记得设置adapter！ 7](#_Toc3422)

[2、 RecyclerView的嵌套，在外部RecyclerView的bindViewHolder中要传入的是内部RecyclerView的数据源而不是RecyclerView 7](#_Toc967)

[ViewPager+Fragment 7](#_Toc19409)

[1、 定制布局 7](#_Toc27924)

[1、 创建Fragment列表 7](#_Toc27543)

[1、创建ViewPager实例 8](#_Toc7638)

[2、onCreate中创建Fragment列表 8](#_Toc16306)

[3、将3个Fragment类实例加入列表 8](#_Toc10883)

[5、绑定ViewPager（findViewById） 8](#_Toc19024)

[ViewPager2 8](#_Toc5596)

[1、 添加依赖 8](#_Toc3920)

[3、 创建adapter 8](#_Toc14299)

[2、 新建一个列表 9](#_Toc17400)

[5、重写getItemCount（）返回列表的大小 9](#_Toc281)

[工具类总结： 9](#_Toc3385)

[BaseQuickAdapter的使用: 11](#_Toc9854)

[5在Activity中使用该适配器 11](#_Toc18742)

[ViewBinding 12](#_Toc24361)

[动画 13](#_Toc1517)

[Interpolator插值器 13](#_Toc28411)

[Glide 14](#_Toc5677)

[Arouter 14](#_Toc18994)

[组件化工程和ARouter 14](#_Toc32578)

[1引入依赖 15](#_Toc17123)

[4在ActivityA中这样跳转到ActivityB 15](#_Toc3327)

[这种情况下就要使用到IProvider 15](#_Toc30562)

[Drawable --shape xml属性详解 17](#_Toc1799)

[LiveDataBus 19](#_Toc12505)

[添加一个广告位： 20](#_Toc6543)

[5、 在onBindViewHolder内添加此广告位的判断 20](#_Toc7065)

[查看设备的qaid 20](#_Toc21736)

[LiveData+ViewModel 20](#_Toc20201)

[Databinding 20](#_Toc652)

[一、单向绑定 20](#_Toc2146)

[2、 可以实现点击确定的时候更改数据源并显示到xml上 21](#_Toc17297)

[二、 双向绑定（以下只写了和单向绑定不同的地方） 21](#_Toc24347)

[1、 在ViewModel中将数据改为liveData类型 21](#_Toc19371)

[依赖无法使用 21](#_Toc5416)

[FindViewById找不到 21](#_Toc25879)

[字体相关 22](#_Toc8775)

[弹窗 22](#_Toc3379)

[4、 在fragment中初始化dialog并调用show方法 22](#_Toc10407)

[广播 22](#_Toc13971)

[3、 实例化TimeChangeReceiver 22](#_Toc19754)

[只要手机启动了就可以监听，程序未启动也可以 23](#_Toc25791)

[3、 调用sentBroadcast（intent）发送广播 23](#_Toc30368)

[小组件 23](#_Toc14096)

[7、 配置清单文件<service name> 23](#_Toc21434)

[PendingIntent 23](#_Toc25407)

[remoteViews.setxxxx() 24](#_Toc17873)

[新建分支 24](#_Toc26023)

[Tab配置 24](#_Toc32439)

[颜色字符串转16进制 25](#_Toc7233)

[天气页（首页）模块添加 25](#_Toc21028)

[1、 编写ModuleStepService，添加跨模块的方法 25](#_Toc27368)

[在setUserVisibleHint中需要更新当前页代码中onCreate中更新的数据 25](#_Toc32643)

[自定义控件 25](#_Toc14959)

[画笔使用 25](#_Toc7975)

[画笔的构造方法（flags） 26](#_Toc29052)

[路径使用 26](#_Toc30259)

[PathMeasure.getPosTan (distance: Float, pos:FloatArray!, tan: FloatArray!) 26](#_Toc15704)

[getTextBounds 26](#_Toc20156)

[RectF 26](#_Toc29193)

[view的绘制原理 27](#_Toc7654)

[Android画笔的三种Style Paint.Style.STROKE 只绘制图形轮廓（描边） Paint.Style.FILL 只绘制图形内容 Paint.Style.FILL\_AND\_STROKE 既绘制轮廓也绘制内容 27](#_Toc17523)

[onSizeChanged方法 29](#_Toc20498)

[Canvas.save()和restore() 29](#_Toc30294)

[getMeasuredWidth和getWidth 30](#_Toc4949)

[2、canvas.drawText() 30](#_Toc2912)

[Text的绘图四线格 30](#_Toc31443)

[渐变 31](#_Toc6064)

[3种详细用法 31](#_Toc27052)

[刷新控件的3种方法（invalidate） 31](#_Toc31602)

[Matrix 32](#_Toc27295)

[LineTo和moveTo 32](#_Toc15500)

[路径path 32](#_Toc409)

[measureText()和getTextBounds() 33](#_Toc30315)

[小组件和通知栏的注意事项 33](#_Toc11962)

[invalidate和postInvalidate的区别及使用 33](#_Toc22893)

[锯齿 33](#_Toc22178)

[来消除锯齿 34](#_Toc16006)

[ImageView的scaleType的属性值 34](#_Toc8705)

[SpannableText啥的 35](#_Toc14600)

[在java文件中加载color资源定义的颜色 35](#_Toc14011)

[创建一个intArray 35](#_Toc8123)

[将颜色字符串转为int值 35](#_Toc32547)

# 开发流程：

1先commit提交到本地，再update更新代码，跑完没问题之后才commit+push

# JSON

JSON分为两种：

JSON对象JSONObject{}， JSON数组JSONArray[]

对应JSON对象字符串，JSON数组字符串

原生：

解析接收来的json字符串比较方便，如果需要打包成json数据则比较繁琐

添加jsonObject数据：

JSONObject jsonObject＝new JSONObject()

jsonObject.put("name", "BOBO")

jsonObject.put("age", 23)

原生转字符串：.toString

代码1：

JSONObject jsonObject = new JSONObject();

jsonObject.put("name", "bobo");

jsonObject.put("age", 100);

String strResult = jsonObject.toString();

Log.d("custom\_json :", strResult);

输出结果为：strResult = {"name":"bobo", "age":"10"};

代码2：

JSONArray jsonArray = new JSONArray();

jsonArray.put("person");

jsonArray.put("books");

jsonArray.put(jsonObject);

String arrStr = jsonArray.toString();

Log.d("custom\_json :", arrStr);

输出结果：

// [

// "person",

// "books",

// {

// "name":"bobo",

// "age":"10"

// }

// ]

可以结合封装好的工具类：

第二个参数可以传入不同类型的值

KeyValueUtils(“key”, “value”)

# FastJSON

# 如何获取recyclerView中点击的view？

通过holder拿到子控件itemView,itemText等然后做判断if itemView=xxx，就执行xxx

在onBindViewHolder中设置点击事件，需要注意的是是通过holder.itemView来点击的，因为itemview你可以把它理解成每一个布局，那么就需要对每一个布局进行点击事件的处理。  
所以我之前写的逻辑是如果itemView不等于xx就执行逻辑不太对，因为itemView肯定会等于xx，应该按文斌写的如果itemView＝xx时候就直接return @onclick这样就不会执行后面的了

# recyclerView的理解

ResyclerView:

ViewHolder的主要任务：容纳View视图。

Adapter从模型层获取数据，然后提供给RecyclerView显示，是沟通的桥梁。

Adapter主要的任务是：创建ViewHolder和将模型层的数据绑定到ViewHolder上。

**position**  
可以理解为list中item的下标

然后在onBindViewHolder方法中写绑定，

通过位置信息和设置来绑定控件和数据

FruitAdapter（val fruitList: List<Fruit>）用于适配布局和数据

1、内部类ViewHolder(View view)的view通常是RecyclerView子项的最外层布局，传入后就可以通过findViewById来绑定布局内的实例了

2、onCreateViewHolder（parent: ViewGroup, viewType: Int）用于创建ViewHolder实例，在这个方法中将item布局加载并传入ViewHolder构造函数内创建一个实例并返回。

3、onBindViewHolder（holder: ViewHolder, position: Int）用于对item的数据进行赋值，会在每个子项滚动到屏幕内的时候执行，可以使用fuitList[position]参数获得fruitList的当前项，再将数据设置到ViewHolder的item布局中即可  
4、getItemCount（）返回数据源的长度即可。

1、Xml添加RecyclerView

2、编写item.xml

Activity

1. 初始化数据fruitList
2. LayoutManager用于指定RecyclerView布局的排列方式
3. 创建Adapter实例传入数据，然后用setAdapter设置RecyclerView的adapter即可

Tips：

1. 记得设置adapter！
2. RecyclerView的嵌套，在外部RecyclerView的bindViewHolder中要传入的是内部RecyclerView的数据源而不是RecyclerView

# ViewPager+Fragment

总思路：

1. 定制布局
2. 创建Fragment类，重写onCreateView来返回Fragment布局
3. 定义Fragment适配器FragmentPagerAdapter管理所有Fragment
4. 通过适配器将Fragment添加到activity页面中的ViewPager中

FragmentPagerAdapter的用法：（已废弃）

FragAdapter extends FragmentPagerAdapter

1. 创建Fragment列表
2. 构造函数FragAdapter（FragmentManager fm, List<Fragment> fragments）接收一个Fragment列表并传递给内部列表
3. 重写getItem获得子Fragment
4. 重写getCount获得子Fragment的数量

现在用的是FragmentStateAdapter

Activity

1、创建ViewPager实例

2、onCreate中创建Fragment列表

3、将3个Fragment类实例加入列表

4、实例化适配器FragAdapter（）传入Manager和Fragment列表，Manager用getSupportFragmentManager（）获得。

5、绑定ViewPager（findViewById）

6、设置adapter（viewPager.setAdapter）

**Activity、FragmentActivity和AppCompatActivity的区别**

1、Activity是最基础的页面类，对应getFragmentManager方法来控制Activity和 Fragment之间的交互。

2、FragmentActivity间接继承自Activity，并提供了对v4包中support Fragment的支 持。在FragmentActivity中必须使用getSupportFragmentManager方法来处理 support Fragment的交互。

3、AppCompatActivity继承自FragmentActivity，为Material Design风格控件提供 了便利。

Tips:

1、Fragment中的onCreateView里View还没创建，要在别的生命周期比如onViewCreated中去写

2、在嵌套fragment中可以用getChildFragmentManager()来获取子fragment的Manager

# ViewPager2

ViewPager2虽然是继承自ViewGroup，但是其实它是**对RecyclerView的封装**

因为ViewPager2内部封装的是RecyclerView，因此它的Adapter也就是recyclerView的Adapter。

**基本使用：**

1. **添加依赖**
2. **将ViewPager2加入布局、并创建需要放进Pager的布局**
3. **创建adapter**
4. **在Activity中为ViewPager设置Adapter**

**Adapter的创建：**

1. **新建MyAdapter继承自RecyclerView.Adapter<MyAdapter.PagerViewHolder>(){**

**}**

1. **新建一个列表**
2. **重写onCreateViewHolder（parent: ViewGroup, viewType: Int）:PagerViewHolder{**

**}在里面通过inflate绑定布局并传入PagerViewHolder（）返回实例**

**4、重写onBindViewHolder（holder: pagerViewHolder, position: Int)在里面调用holder.bindData（mList[position]）绑定数据**

5、重写getItemCount（）返回列表的大小

6、新建内部类PagerViewHolder(itemView: View): RecyclerView.ViewHolder(itemView){}在里面绑定子item的布局并设置UI

7、

Activity中设置adapter:

1. 新建ViewPager实例并绑定布局
2. 新建adapter实例并设置adapter

Tips：

设置竖直滑动：viewPager2.orientation = ViewPager2.ORIENTATION\_VERTICAL

# 工具类总结：

**对于具体的使用可以看Utils源码也可以在全局搜索对应的工具来看别人如何使用的**

**获取上下文工具类：**

ContextUtils.get.context

**键值对存储工具类（底层通过MMKV实现）：**

直接调用setxxx就可以存储键值对

KeyValueUtils.setLong(KEY\_FIRST\_TIME\_LAUNCH\_APP, System.currentTimeMillis())

**动画工具类：**

看起来好像是给个String就可以加载对应的动画

ImageLoaderUtils.loadImageOrGifOrJson(Context context, LottieAnimationView imageView, String imageUrl)

**日志工具类：**

LogUtils.e("AbValueManager", "获取Ab接口失败: $msg")

**网络工具类：**

post请求和get请求里面都有

NetHelper.post(NetParams.getUrl(url)) //

**线程工具类：**

// 主线程

ThreadUtils.runInUiThread(new Runnable() {

@Override

public void run() {

Log.d(TAG, "onDone: 主线程执行");

}

});

ThreadUtils.runInUiThreadDelay(() -> {

Log.d(TAG, "onDone: 主线程延迟执行");

}, 2000);

//子线程

ThreadUtils.runInWorkThread(() -> {

Log.d(TAG, "onDone: 子线程执行");

});

ThreadUtils.runInWorkThreadDelay(() -> {

Log.d(TAG, "onDone: 子线程延迟执行");

}, 2000);

**图片加载工具类：**

底层通过Glide实现

ImageLoaderUtils

**Gson工具类：**

GsonUtil.toJson(Object)

GsonUtil.fromJson(String s, Class<T> tClass)

**天气工具类：**

WeatherUtils.getWeatherWarningType(item.code)

提供了很多对应的转换比如通过接口传来的code获得type等等

**日历工具类：**

CalendarUtil

LocalDate

LocalTime

CityDisplayModel

(注意导的包是第三方的)

**状态栏工具类：**

StatusBarUtil

**应用程序工具类:**

Utils

可以调用getApp获取当前应用程序的实例

**资源工具类：**

ResourceUtils

(int) ResourceUtils.getDimension(R.dimen.base\_dp\_20, 0F);

可以调用当前系统的dp来使用

# BaseQuickAdapter的使用:

BaseQuickAdapter<T, VH : BaseViewHolder>

T:在类/接口内部使用统一的数据类型就是ItemBean,可以传入 T 及其子类类型

VH : 对应ViewHolder，一般直接填BaseViewHolder就可以，限制形参不为 null 需要手动指定为 <T : Any>

使用BaseQuickAdapter：

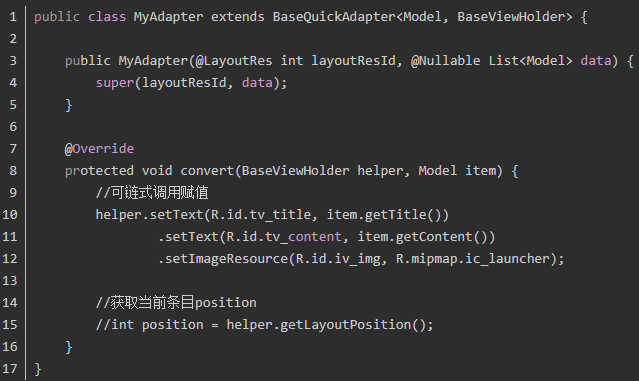
1在xml添加recyclerView

2编写item.xml

3编写对应的实体Model.java\

4编写适配器

//绑定xml和实体类

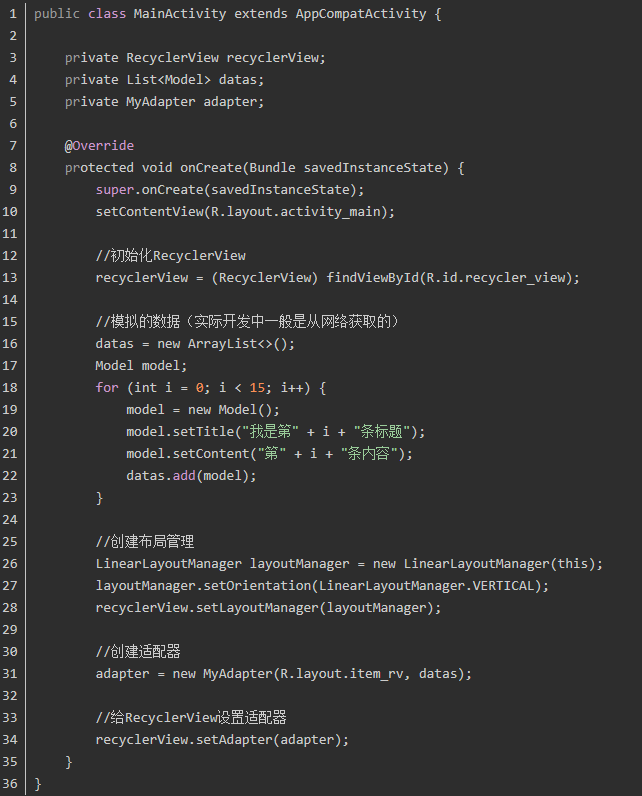


如果是加载网络图片或自定义view可以通过viewHolder.getView(viewId)获取该控件。

5在Activity中使用该适配器

//创建布局管理，给recyclerView设置布局管理，

创建适配器，给recyclerView设置适配器



# ViewBinding

# 动画

1．逐帧动画(frame-by-frame animation) : 几张图片进行快速的播放形成的动画

2．补间动画(tweened animation)：设置初始值、结束值。中间动态由安卓完成

3．属性动画(property animation)：对组件的属性进行改变，从而达到动画的效果

主要通过 **ValueAnimator**或**ObjectAnimator**，然后可选择性地配合[监听器](https://so.csdn.net/so/search?q=%E7%9B%91%E5%90%AC%E5%99%A8&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_52551903/article/details/_blank)完成。

使用LottieAnimationView来设置动画

## Interpolator插值器

在实现动画的非线性变化的方法中，常用的一种是为动画添加插值器以改变视图的属性值，从而实现理想的动画效果。Interpolator使用相对简单

angleAnim.setInterpolator(new AccelerateDecelerateInterpolator());

插值器种类：

[(3条消息) 几种常用的Interpolator（插值器）的动画效果\_setinterpolator\_ionetwogo的博客-CSDN博客](https://blog.csdn.net/qq_15128547/article/details/50779964?ops_request_misc=%7B%22request%5Fid%22%3A%22168180012116800182166467%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=168180012116800182166467&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-50779964-null-null.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&utm_term=setInterpolator&spm=1018.2226.3001.4187&ydreferer=aHR0cHM6Ly9zby5jc2RuLm5ldC9zby9zZWFyY2g/cT1zZXRJbnRlcnBvbGF0b3ImdD0mdT0mdXJ3PQ==&ydreferer=aHR0cHM6Ly9zby5jc2RuLm5ldC9zby9zZWFyY2g/cT1zZXRJbnRlcnBvbGF0b3ImdD0mdT0mdXJ3PQ==)

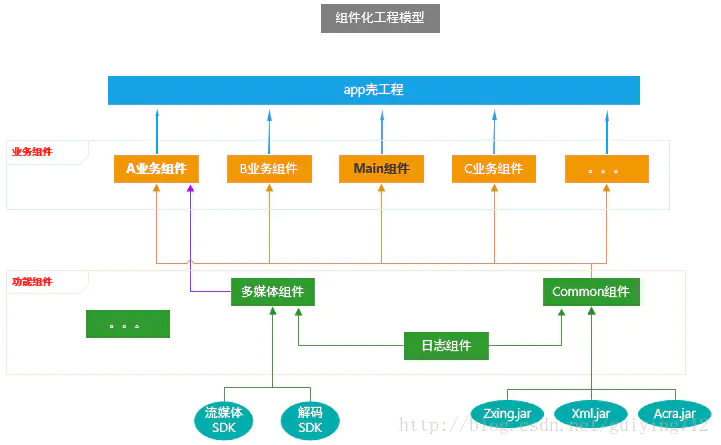
<LottieAnimationView>

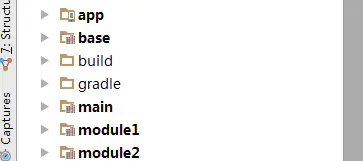
1. 在json里面配置
2. 在xml中直接引用
3. 也可以动态加载

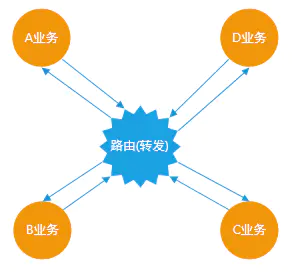
# Glide

# Arouter

# 组件化工程和ARouter







这是组件化工程模型下的业务关系，业务之间将不再直接引用和依赖，而是通过“路由”这样一个中转站间接产生联系，我们今天要讲的就是这个路由。

假如ActivityA想跳转到ActivityB怎么实现呢？通常跳转是这么写的：

Intent intent = new Intent(this,ActivityB.class);

startActivity(intent);

但是现在业务A和业务B在不同的moudle中，A业务中没法 import ActivityB，因为A和B是相互独立，没有依赖的。  
路由做的事情就是在不用引导类的情况下实现这一跳转。

1引入依赖

2（建议）在Application中初始化

3使用注解标记要跳转的Activity@Route(path = "/module\_b/activity\_b")

4在ActivityA中这样跳转到ActivityB

*ARoute*r.getInstance().build("/module\_b/activity\_b")

.withString("key3", "888")

.withObject("key4", new Test("Jack", "Rose"))

.navigation();

在ActivityB中获取参数有两种方式，一种是普通Activity那样getIntent().getXXX  
另一种是使用@Autowired注解的方式

@Autowired

*pu*blic String name;

// ARouter会自动对字段进行赋值，无需主动获取

*Lo*g.d("param", name);

还有一种情况，业务A是要使用业务C中的某个服务或者说某种功能

这种情况下就要使用到IProvider



申明的接口放在base module中，接口的实现放在具体功能实现的module中。

然后在要使用这一服务的module中这样使用：

HelloService helloService = (HelloService) ARouter.getInstance().build("/service/hello").navigation();

helloService.sayHello("Vergil");

这样就实现了调用其他业务服务

还有还可能通过ARouter获取其他业务模块的Fragment:

// 获取Fragment

Fragment fragment = (Fragment) ARouter.getInstance().build("/test/fragment").navigation();

@Autowired

ARouter 有一个@Autowired 的注解，能自动帮我们赋值一些变量通过Arouter.getInstance().[inject](https://so.csdn.net/so/search?q=inject&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u010648159/article/details/_blank)(this);就能将 Activity 传输的一些值自动帮我们赋值上对应变量，省去我们手动去调用getIntent().getString(xxx)

//跳转

ARouter.getInstance()

                    .build(IGlobalRoutePathConsts.CITY\_LOCATION\_PAGE)

                    .withString("cityName", mCityName)

                    .withString("cityCode", mCityCode)

                    .withString("cityDesc", "")

                    .withBoolean("isPositioning", false)

                    .navigation()

//声明 Activity#onCreate() 中初始化

@Route(path = IGlobalRoutePathConsts.CALENDAR\_PAGE\_ACTIVITY)

class CalendarActivity : BaseLoadingActivity() {

    @Autowired

    @JvmField

    var open\_entrance = "" //来源

    override fun onCreate(savedInstanceState: Bundle?) {

        ARouter.getInstance().inject(this);

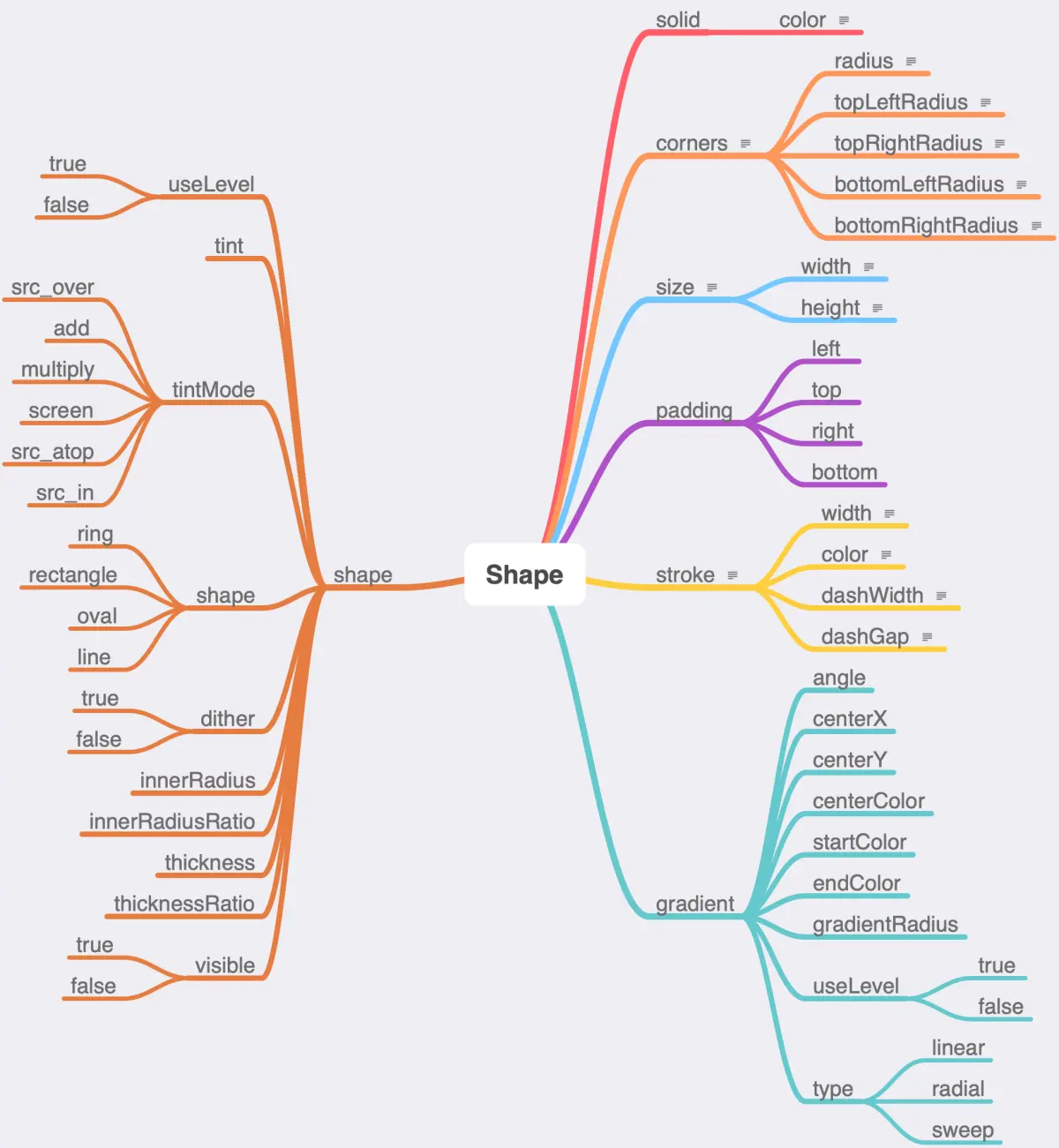
        LogUtils.d("Don", "打印来源=${open\_entrance}")

    }

}

# Drawable --shape xml属性详解

<https://www.jianshu.com/p/f424779bd40f>



左侧是shape标签的自身属性，右侧是shape标签所支持的标签和标签的属性。

属性说明:

<?xml version="1.0" encoding="utf-8"?>

<shape xmlns:android="http://schemas.android.com/apk/res/android"

android:dither=["true" | "false"] //将在位图的像素配置与屏幕不同时（例如：ARGB 8888 位图和 RGB 565 屏幕）启用位图的抖动；值为“false”时则停用抖动。默认值为 true。

android:shape=["rectangle" | "oval" | "line" | "ring"]//分别为矩形、椭圆、线、环。默认为矩形rectangle

android:innerRadius="integer" // shape为ring时有效，内环半径

android:innerRadiusRatio="float" // shape为ring时有效，内环的厚度比，即环的图形宽度与内环半径的比例，按照这个比例计算内环半径，默认为3，可被innerRadius值覆盖

android:thickness="integer" // shape为ring时有效，环的厚度

android:thicknessRatio="float" // shape为ring时有效，环的厚度比，即环的图形宽度与环的厚度的比例，按照这个比例计算环的厚度，默认为9，可被thickness值覆盖

android:tint="color" // 给shape着色

android:tintMode=["src\_in" | "src\_atop" | "src\_over" | "add" | "multiply" | "screen"] // 着色类型

android:useLevel=["true" | "false"] // 较少用，一般设为false，否则图形不显示。为true时可在LevelListDrawable使用

android:visible=["true" | "false"] >

<!-- 圆角 -->

<corners

android:radius="integer" // 圆角半径，设置下面四个属性时，对应的位置属性会被覆盖

android:topLeftRadius="integer" // 左上角圆角半径

android:topRightRadius="integer" // 右上角圆角半径

android:bottomLeftRadius="integer" // 左下角圆角半径

android:bottomRightRadius="integer" // 右下角圆角半径

/>

<!-- 渐变 -->

<gradient

android:type=["linear" | "radial" | "sweep"]// 渐变类型，线性、放射性、扫描性；默认为线性

android:angle="integer" // 渐变角度，渐变类型为linear时有效；默认为0，从左至右渐变，角度逆时针方向计算，角度需要时45的整数倍数

android:centerColor="integer" // 渐变中间位置颜色

android:startColor="color" // 渐变开始位置颜色

android:endColor="color" // 渐变结束位置颜色

android:centerX="float" // 设置渐变中心的X坐标，取值区间[0,1]，默认为0.5，即中心位置

android:centerY="float" // 设置渐变中心的Y坐标，取值区间[0,1]，默认为0.5，即中心位置

android:gradientRadius="integer" // type为放射性渐变radial时有效，渐变的半径

android:useLevel=["true" | "false"] // 与shape中该属性的一致

/>

<!-- 内边距 -->

<padding

android:left="integer" // 左边距

android:top="integer" // 上边距

android:right="integer" // 右边距

android:bottom="integer" // 下边距

/>

<!-- 大小 -->

<size

android:width="integer" // 图形宽度

android:height="integer" // 图形高度

/>

<!-- 填充 -->

<solid

//"#1AE7E7E7" 表示10%的E7E7E7,1A在十六进制中表示10%

android:color="color" // 图形的填充色 />

<!-- 描边 -->

<stroke

android:width="integer" // 描边的宽度

android:color="color" // 描边的颜色

android:dashWidth="integer" // 虚线宽度

android:dashGap="integer" // 虚线间隔

/>

</shape>

# LiveDataBus

# 添加一个广告位：

1. 找到对应的页面的adapter和里面的list内广告（ADP）的位置（一般可以直接搜索到模块对应的名称的位置）
2. 在IAdPosConsts内新增广告位id
3. 将对应的广告id位添加进list内
4. 在onCreateViewHolder内添加此广告id位的判断
5. 在onBindViewHolder内添加此广告位的判断
6. 以防万一，进入Holder（BaseInsertADHolder）内查看广告类型和判断是否正确

# LiveData+ViewModel

1. 新建ViewModel并将里面的变量写为MutableLiveData<>
2. 在activity中创建ViewModel实例（ViewModelProvider().get()）
3. 绑定控件并在登录点击事件中调用mainViewModel.account.setValue(etAccount.getText().toString().trim())获取xml中的值并设置到ViewModel实例中（mainViewModel.account.setValue，setValue是liveData中特有的方法）
4. 调用.observe()方法对ViewModel中的数据进行观察并在改变时通知页面最新的值

Tips：

1、可以实现观察ViewModel中的数据并在改变时改变页面数据

# Databinding

一、单向绑定

1. 新建一个Use类并在set方法中调用notifyChange（）方法通知改变

在xml中和User类绑定并在需要修改的属性中的text上写android:text="@{user.pwd}"

1. 在activity中新建User对象、绑定xml（dataBinding）
2. 调用dataBinding.btnLogin可以实现监控xml中的数据改变：

user.setAccount(dataBinding.etAccount.getText().toString().trim())（括号内的dataBinding用来监控xml页面的数据并获取之后给user对象，user已经绑定了xml，所以会直接改变）

Tips:

1. dataBinding不需要去手动的findViewById了，系统会自动生成驼峰命名对象（dataBinding.etAccount）
2. 可以实现点击确定的时候更改数据源并显示到xml上
3. 双向绑定（以下只写了和单向绑定不同的地方）
4. 在ViewModel中将数据改为liveData类型
5. 在User类中将notifyChange修改成notifyPropertyChanged()只通知对应属性改变
6. ‘[在xml中绑定的User类改为绑定ViewModel并在需要赋值的地方改为@={viewModel.user.account}，表示直接显示当前的数据](mailto:在xml中绑定的User类改为绑定ViewModel并在需要赋值的地方改为@=%7bviewModel.user.account%7d，表示直接显示当前的数据)
7. 实例化ViewModel
8. 获取观察对象：获取liveData并观察



Tips：

1. 好像就是dataBinding的基础上加上了livedata+ViewModel
2. 可以实现在页面上写数据的同时数据源改变并同步显示到xml中

# 依赖无法使用

去build.gradle内看看是不是添加了什么依赖，可以把implement改为api，这样就把依赖给扩大到所有的包都可以使用了

# 直接用id引用和使用binding？

Kotlin可以直接用id名引用的猪，另外在

# 字体相关

1. 添加字体到assets-fonts文件夹下
2. 使用自带的CustomFontTextView
3. 属性内使用app:appTypeFace="@string/font\_akrobat"

Tips：

1. 设置不了居中，只能不限制高度
2. 设置不了加粗

# 弹窗

1. 编写弹窗页面
2. 编写弹窗样式style（在value内）
3. 新建ClockAlertDialog继承自AppCompatDialog并在中间加载样式、布局、点击事件等（可全局搜索ClockAlertDialog看代码）
4. 在fragment中初始化dialog并调用show方法

# 广播

标准广播：全局异步

有序广播：同步，有接收优先级

动态注册接受者：

可以自由的控制注册与注销，只有程序启动之后才能接收广播

1. 创建广播接收者：新建类TimeChangeReceiver继承自BroadcastReceiver，并重写onReceive方法在里面实现需要的逻辑
2. 在activity中创建过滤器IntentFilter并addAction添加需要监听的广播
3. 实例化TimeChangeReceiver
4. 调用registerReceiver()方法将接收者和过滤器都传入注册
5. 在onDestroy（）中取消注册unregisterReceiver（timeChangeReceiver）

静态注册接收者：

只要手机启动了就可以监听，程序未启动也可以

1. 创建广播接收者：新建类TimeChangeReceiver继承自BroadcastReceiver，并重写onReceive方法在里面实现需要的逻辑
2. 在注册文件中用<receiver>进行注册，并使用<user-permission>声明权限（如果需要的话）

发送自定义广播：

1. 在activity中使用intent(“com.example.broadcasttest.MY\_BROADCAST”)
2. 调用intent.setPackage(packageName)指定发送给的程序包名（静态注册广播无法接收隐式广播，将其变为显式）
3. 调用sentBroadcast（intent）发送广播

Tips：

// 从Android9.0开始，系统不再支持静态广播，应用广播只能通过动态注册

# 小组件

1. 创建监视器WightProvider继承于AppWidgetProvider并重写onDeleted、onDisabled、onEnabled、onReceive和onUpdate
2. 创建小组件布局
3. 创建小组件配置文件，绑定小组件布局、设置宽高、刷新时间
4. 在清单文件内配置
5. <receiver name:>

<intent-filter>

<action name:>

<meta-data

name

Resource

1. 创建服务TimeService继承于Service来编写逻辑
2. 配置清单文件<service name>
3. 在监视器的onEnabled（添加到桌面）时开启服务(startService)，onDisabled（移除桌面）时关闭服务

# PendingIntent

PendingIntent可以看作是对Intent的一个封装，但它不是立刻执行某个行为，

而是满足某些条件或触发某些事件后才执行指定的行为。

PendingIntent的获取

PendingIntent获取有三种方式:通过Activity,Service,BroadcastReceiver获取.

1. 你可以通过getActivity(Context context, int requestCode, Intent intent, int flags)系列方法从系统 取得一个用于启动一个Activity的PendingIntent对象.

2.可以通过getService(Context context, int requestCode, Intent intent, int flags)方法从系统取得一个 用于启动一个Service的PendingIntent对象.

3.可以通过getBroadcast(Context context, int requestCode, Intent intent, int flags)方法从系统取得一个用于向BroadcastReceiver的发送广播的PendingIntent对象.

# remoteViews.setxxxx()

如果它有提供的就可以直接用，没有的话就用setInt

remoteViews.setInt(

R.id.container,"setBackgroundResource",WeatherUtils.getWeatherSolarTerms4x3BackgroundIdByLocalDate(

LocalDate.now()

)

)

原理：用反射获得此控件的方法名再调用

# 新建一个分支之后该如何运行

先导入依赖再切换变体

# Tab配置

1. 在router表中添加路径
2. 在IMainConsts添加id
3. 在MainViewModel中添加判断
4. 在需要跳转的类上方打上标注@Route(path = IGlobalRoutePathConsts.FRAGMENT\_WEATHER\_15DAY\_PAGE)

# 颜色Color字符串转16进制

Color.parseColor("#ff123456");

# 天气页（首页）模块添加

1. 编写ModuleStepService，添加跨模块的方法
2. 在cityWeatherAdapter中的onCreateViewHolder中调用跨模块方法创建
3. 只要Holder是继承自baseHolder的就不需要编写onbind那个方法了因为做了自动化，会自动绑定了。

# 在setUserVisibleHint中需要更新当前页代码中onCreate中更新的数据

因为第二次点到页面时调用setUserVisibleHint不会调用onCreate

# 自定义控件

如果我想在自定义控件外面去判断修改他的文字可能实现吗？

可以，还有一个方法是直接在java文件中加载自定义控件。

自定义控件，类型各有各的分法，本人的对其进行大概如下分类：  
\* 自制控件 ： 该自定义控件继承自View或者ViewGroup，自己绘制。  
\* 组合控件 ： 利用系统已提供的控件，组合成一个新的控件  
\* 拓展控件 ： 继承自系统已提供的控件并且加上新的功能或者特性。

Offset：偏移量

## 画笔使用

this.mTextPaint.setStrokeWidth(2f);

this.mTextPaint.setStyle(Paint.Style.FILL\_AND\_STROKE);

设置字体画笔的宽度，这两个似乎要一起用。

[(3条消息) Paint（一）：画笔基本使用\_qq\_33393910的博客-CSDN博客](https://blog.csdn.net/qq_33393910/article/details/81185249?ops_request_misc=%7B%22request%5Fid%22%3A%22168178203816782427419681%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=168178203816782427419681&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-81185249-null-null.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&utm_term=Paint&spm=1018.2226.3001.4187)

## 画笔的构造方法（flags）

[(3条消息) Android Paint应用一\_anti\_alias\_flag\_洛蛮帝的博客-CSDN博客](https://blog.csdn.net/lp598876762/article/details/93982889?ops_request_misc=%7B%22request%5Fid%22%3A%22168178226516800217228583%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=168178226516800217228583&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-93982889-null-null.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&utm_term=Paint.ANTI_ALIAS_FLAG&spm=1018.2226.3001.4187)

## 路径使用

[(6条消息) Android Path 使用 解析\_android path用法\_七夜君君的博客-CSDN博客](https://blog.csdn.net/qq2582494/article/details/83897261?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%AE%89%E5%8D%93path&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-4-83897261.142%5ev86%5einsert_down28v1,239%5ev2%5einsert_chatgpt&spm=1018.2226.3001.4187)

SetPath：

[(6条消息) Android基础之Path类的使用\_Exploring1024的博客-CSDN博客](https://blog.csdn.net/Mr_dsw/article/details/48931515?ops_request_misc=%7B%22request%5Fid%22%3A%22168249244116782425152055%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=168249244116782425152055&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_v2~default-4-48931515-null-null.142%5ev86%5einsert_down28v1,239%5ev2%5einsert_chatgpt&utm_term=%E5%AE%89%E5%8D%93path&spm=1018.2226.3001.4187)

## PathMeasure.getPosTan (distance: Float, pos:FloatArray!, tan: FloatArray!)

getPosTan 函数 作用是 获取指定长度位置的坐标点 以及该点对应切线的值 ;

* **distance: Float 参数** : 沿当前轮廓进行采样的距离 ;
* **pos: FloatArray! 参数** : 如果不为null，则返回采样位置（x==[0]，y==[1]） ;
* **tan: FloatArray! 参数** : 如果不为null，则返回采样的切线（x==[0]，y==[1]）;

tan的切线是返回切线上的一点（与pos连接即为切线）

## getTextBounds

getTextBounds(String text,int start,int end,Rext bounds),它的**中文解释**是：返回一个包含中文的矩形边界，位置为(start,end)

## RectF

RectF和Rect都表示的是一个矩形的区域，他们的构造方法类似

RectF 中的参数是单精度浮点型的；  
Rect 中的参数是整形的。

//设置矩形（rectangle）

val rectF = RectF(

(index \* itemWidth).toFloat(),

viewBottom + PxUtils.dip2px(12f).toFloat(),

PxUtils.dip2px(72f).toFloat() + index \* itemWidth,

viewBottom + PxUtils.dip2px(12f).toFloat() + PxUtils.dip2px(18f).toFloat(),

)

canvas?.drawRoundRect(rectF, 10f, 10f, probabilityPaint)

//绘制圆角矩形

DrawRect//绘制矩形

DrawArc//绘制圆弧

## view的绘制原理

1.measure用来测量View的宽和高。  
2.layout用来确定View在父容器中放置的位置。  
3.draw用来将view绘制在屏幕上

## Android画笔的三种Style Paint.Style.STROKE 只绘制图形轮廓（描边） Paint.Style.FILL 只绘制图形内容 Paint.Style.FILL\_AND\_STROKE 既绘制轮廓也绘制内容

PathEffect在Java中是一个父类，就是感觉drawPath的时候的效果

他有以下子类

CornerPathEffect DashPathEffect  DiscretePathEffect PathDashPathEffect ComposePathEffect SumPathEffect

**CornerPathEffect：**

这个类的作用就是将Path的各个连接线段之间的夹角用一种更平滑的方式连接，类似于圆弧与切线的效果。  
一般的，通过CornerPathEffect(float radius)指定一个具体的圆弧半径来实例化一个CornerPathEffect。

**DashPathEffect：**

这个类的作用就是将Path的线段虚线化。  
构造函数为DashPathEffect(float[] intervals, float offset)，其中intervals为虚线的ON和OFF数组，该数组的length必须大于等于2，phase为绘制时的偏移量。

代码中的float数组,必须是偶数长度,且>=2,指定了多少长度的实线之后再画多少长度的空白.

如本代码中,绘制长度1的实线,再绘制长度2的空白,再绘制长度4的实线,再绘制长度8的空白,依次重复

**DiscretePathEffect：**

这个类的作用是打散Path的线段，使得在原来路径的基础上发生打散效果。  
一般的，通过构造DiscretePathEffect(float segmentLength,float deviation)来构造一个实例，其中，segmentLength指定最大的段长，deviation指定偏离量。

**PathDashPathEffect：**

这个类的作用是使用Path图形来填充当前的路径，其构造函数为PathDashPathEffect (Path shape, float advance, float phase,PathDashPathEffect.Stylestyle)。  
shape则是指填充图形，advance指每个图形间的间距，phase为绘制时的偏移量，style为该类自由的枚举值，有三种情况：Style.ROTATE、Style.MORPH和  
Style.TRANSLATE。其中ROTATE的情况下，线段连接处的图形转换以旋转到与下一段移动方向相一致的角度进行转转，MORPH时图形会以发生拉伸或压缩等变形的情况与下一段相连接，TRANSLATE时，图形会以位置平移的方式与下一段相连接。

**ComposePathEffect：**

组合效果，这个类需要两个PathEffect参数来构造一个实例，ComposePathEffect (PathEffect outerpe,PathEffect innerpe)，表现时，会首先将innerpe表现出来，然后再在innerpe的基础上去增加outerpe的效果。

**SumPathEffect：**

叠加效果，这个类也需要两个PathEffect作为参数SumPathEffect(PathEffect first,PathEffect second)，但与ComposePathEffect不同的是，在表现时，会分别对两个参数的效果各自独立进行表现，然后将两个效果简单的重叠在一起显示出来。

**关于参数phase**

在存在phase参数的两个类里，如果phase参数的值不停发生改变，那么所绘制的图形也会随着偏移量而不断的发生变动，这个时候，看起来这条线就像动起来了一样。phase感觉就是把图像平移phase段距离，他会每次都会调用onDraw方法，重而实现看起来在动的方法

### onMeasure方法

* 1.onMeasure方法会在view的位置信息发生变化时调用。
* 2.子view回调了onMeasure方法，父view一定也会回调onMeasure方法。
* 3.onMeasure方法回调后，一定会回调onLayout方法,之后一定会调用ondraw方法。 这个就是 测量,排版,绘制.

## onSizeChanged方法

在控件大小发生改变时调用。

作用：获取控件的宽和高度

## Canvas.save()和restore()

save()和restore()方法是绘制复杂图形必不可少的方法.它们分别是用来保存和恢复 canvas 状态的，都没有参数。

canvas.save()用来保存先前状态的

canvas.restore()用来恢复之前保存的状态

注：两种方法必须搭配使用，否则没有效果

对canvas中特定元素的旋转平移等操作实际上是对整个画布进行了操作， 默认情况下那么每一次绘图都会在上一次的基础上进行操作，最后导致错位。

比如 页面上的元素相对于原点30度递增旋转，30，60，90.如果不使用save 以及 restore

就会变成30, 90, 150，每一次在前一次基础上进行了旋转。

## getMeasuredWidth和getWidth

getMeasuredWidth方法和getWidth方法都是View类中的方法，用于获取控件的宽度信息。但是它们的含义以及使用时机略有不同。

getMeasuredWidth方法返回的是控件在最后一次测量时所得到的宽度数值。这个宽度数值没有被任何布局参数所修改（如weight、match\_parent等），也没有考虑控件的本身大小限制，因此并不一定等于控件实际的显示宽度。通常情况下，getMeasuredWidth方法只在自定义View的onMeasure方法中被调用，用于获取子View的实际测量宽度。

getWidth方法则返回的是控件当前实际显示的宽度数值，即经过布局参数处理和大小限制之后的实际宽度值。在控件已经经过测量、布局等过程，并成功显示在屏幕上之后，可以通过getWidth方法获取到其实际宽度值。需要注意的是，如果该控件还没有完成渲染或者还未添加到父布局中，则该方法返回的值为0。

总之，getMeasuredWidth方法主要用于测量阶段获取控件的测量宽度，getWidth方法则用于在渲染完成后获取控件的实际宽度。

## 2、canvas.drawText()

/\*\*

\* text:要绘制的文字

\* x：绘制原点x坐标

\* y：绘制原点y坐标

\* paint:用来做画的画笔

\*/

public void drawText(String text, float x, float y, Paint paint)

注意：

一般而言，(x，y)所代表的位置是所画图形对应的矩形的左上角点。但在drawText中是非常例外的，y所代表的是基线的位置！

## Text的绘图四线格

前面我们讲了基线，其实除了基线，系统在绘制Text时，还是有其它线的，我们来看个图：



除了基线以外，如上图所示，另外还有四条线，分别是ascent,descent,top,bottom，他们的意义分别是：

ascent: 系统建议的，绘制单个字符时，字符应当的最高高度所在线

descent:系统建议的，绘制单个字符时，字符应当的最低高度所在线

top: 可绘制的最高高度所在线

bottom: 可绘制的最低高度所在线

## 渐变

Android 支持三种颜色渐变， LinearGradient（线性渐变）、 RadialGradient （环形渐变）、 SweepGradient（扫描渐变）。

这三种渐变均继承自android.graphics.Shader， Paint 类通过setShader()方法来支持渐变。

paint.shader = LinearGradient(

itemViewList[0].pointFMax.x,

tempHeight.toFloat(),

itemViewList[0].pointFMax.x,

viewHeight.toFloat(),

Color.parseColor("#44FF6767"),

Color.parseColor("#00000000"),

Shader.TileMode.CLAMP

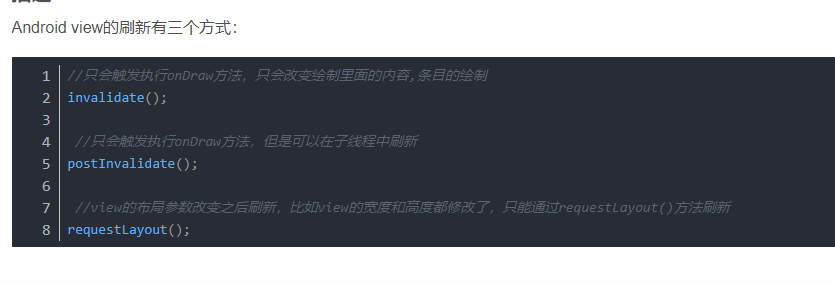
)

3种详细用法

[(3条消息) Android View — Gradient 渐变\_android渐变view\_itzilong的博客-CSDN博客](https://blog.csdn.net/zhizhuodewo6/article/details/80406018?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%AE%89%E5%8D%93%E6%B8%90%E5%8F%98&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-80406018.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&spm=1018.2226.3001.4187)

## 刷新控件的3种方法（invalidate）

在自定义控件已完成绘制后修改属性的话需要调用刷新控件。



## Matrix

**Matrix是一个矩阵，主要功能是坐标映射，数值转换。**我们的内容区和屏幕坐标系还相差一个通知栏加一个标题栏的距离，所以两者是不重合的

更加详细的原理：

[(3条消息) Matrix简介\_weixin\_34273479的博客-CSDN博客](https://blog.csdn.net/weixin_34273479/article/details/91863273?ops_request_misc=&request_id=&biz_id=102&utm_term=Matrix&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-91863273.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&spm=1018.2226.3001.4187)

## LineTo和moveTo

这里可以看到moveTo，是把笔尖moveTo到指定位置，然后使用lineTo去画。

而lineTo是直接画到指定的位置。当没有设置moveTo的时候，那么这个位置就为空，这样话lineTo的第一个点就没有东西，就是第一个图。

## 路径path

先设置画笔，再设置路径，最后用drawPath将路径画出来

this.path1.lineTo(itemViewList[0].pointFMax.x, viewHeight.toFloat())

this.path1.lineTo(itemViewList[0].pointFMax.x, itemViewList[0].pointFMax.y)

canvas.drawPath(path1, this.paint)

贝塞尔曲线

二次贝塞尔曲线对应

quadTo(float x1,float y1,float x2, float y2) ,

三次贝塞尔曲线对应 cubicTo(float x1,float y1, float x2, float y2, float x3,float y3)

构造方法和基本方法：

[Path相关方法讲解(一)\_path.reset()\_GAStudio的博客-CSDN博客](https://blog.csdn.net/tianjian4592/article/details/45652257?ops_request_misc=%7B%22request%5Fid%22%3A%22168180342616800225516355%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=168180342616800225516355&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-45652257-null-null.142%5ev84%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&utm_term=path.reset&spm=1018.2226.3001.4187&ydreferer=aHR0cHM6Ly9zby5jc2RuLm5ldC9zby9zZWFyY2g/cT1wYXRoLnJlc2V0JnQ9JnU9JnVydz0=)

XXXTo 和 addXXX 和贝塞尔曲线：

[Path相关方法讲解(二)\_GAStudio的博客-CSDN博客](https://blog.csdn.net/tianjian4592/article/details/46955833)

## measureText()和getTextBounds()

两种测量方式为什么不同（参考网上）：

measureText() 会在文本的左右两侧加上一些额外的宽度，这部分额外的宽度叫做 Glyph’s AdvanceX （具体应该是属于字型方面的范畴，我猜测这部分宽度是类似字间距之类的东西）

getTextBounds() 返回的则是当前文本所需要的最小宽度，也就是整个文本外切矩形的宽度，而且两个函数的精度也是不同的。

# 小组件和通知栏的注意事项

只能用最基础的四大布局（线性布局和相对布局）

嵌套最多3层

# invalidate和postInvalidate的区别及使用

相同点：都是用来更新当前的View

invalidate在主线程中调用，postInvalidate在子线程中调用，最终调的也是view.invalidate()

# 锯齿

在使用画笔自定义绘制线段和多边形时，缩放斜边会出现锯齿，  
如果是一条斜着的线用canvas画时，就会出现锯齿效果，这是由于像素点的问题造成的

可以用canvas.setDrawFilter(new PaintFlagsDrawFilter(0, Paint.ANTI\_ALIAS\_FLAG | Paint.FILTER\_BITMAP\_FLAG));

canvas.setDrawFilter(pfd);

来消除锯齿

# ImageView的scaleType的属性值

[MATRIX](https://so.csdn.net/so/search?q=MATRIX&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_34902522/article/details/_blank),FIT\_XY,FIT\_START,FIT\_END,FIT\_CENTER,CENTER,CENTER\_CROP,CENTER\_INSIDE.

1.android:scaleType=“center”  
保持原图的大小，显示在ImageView的中心。当原图的size大于ImageView的size时，多出来的部分被截掉。  
2.android:scaleType=“center\_inside”  
以原图正常显示为目的，如果原图大小大于ImageView的size，就按照比例缩小原图的宽高，居中显示在ImageView中。如果原图size小于ImageView的size，则不做处理居中显示图片。  
3.android:scaleType=“center\_crop”  
以原图填满ImageView为目的，如果原图size大于ImageView的size，则与center\_inside一样，按比例缩小，居中显示在ImageView上。如果原图size小于ImageView的size，则按比例拉升原图的宽和高，填充ImageView居中显示。  
4.android:scaleType=“matrix”  
不改变原图的大小，从ImageView的左上角开始绘制，超出部分做剪切处理。  
5.androd:scaleType=“fit\_xy”  
把图片按照指定的大小在ImageView中显示，拉伸显示图片，不保持原比例，填满ImageView.  
6.android:scaleType=“fit\_start”  
把原图按照比例放大缩小到ImageView的高度，显示在ImageView的start（前部/上部）。  
7.android:sacleType=“fit\_center”  
把原图按照比例放大缩小到ImageView的高度，显示在ImageView的center（中部/居中显示）。  
8.android:scaleType=“fit\_end”  
把原图按照比例放大缩小到ImageView的高度，显示在ImageVIew的end（后部/尾部/底部）

# SpannableText啥的

可以用来控制文字的显示方式、更强大的可以控制中间几个字哪几个是什么样子的

# 在java文件中加载color资源定义的颜色

getResources().getColor(R.color.summer\_airquality\_dafault\_paint\_color)

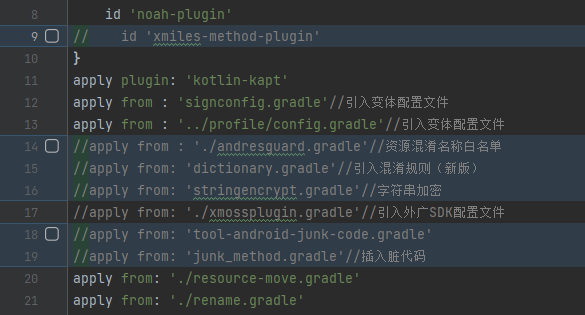
# 创建一个intArray

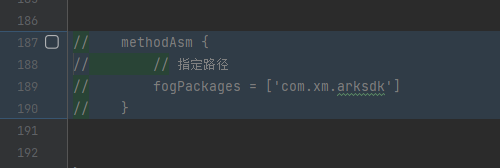
*va*l intArrayOf = intArrayOf(1, 3, 4)

# 将颜色字符串转为int值

Color.parseColor("#00F5F4FF")

# 项目加载缓慢可以注释的代码（切记不能提交！！！）





修改之后记得sync 一下