**Video项目的总结**

1. **View高度和宽度的获取**

**首先getMeasuredWidth()和getLayoutParams().width获取的是view原始的大小，也就是这个view在XML文件中配置或者是代码中设置的大小。getWidth()获取的是这个view最终显示的大小，这个大小有可能等于原始的大小也有可能不等于原始大小。getMeasuredWidth()在onMeasure()执行完后才会有值 。**

**但是，项目中有时用getMeasuredWidth()和getWidth()这两个方法去获取view的长度时，获取到的确始终为0。**

**这是因为每个布局都要经过如下三个步骤:**

**测量：onMeasure 设置自己显示在屏幕上的宽高   
布局：onLayout 设置自己显示在屏幕上的位置(只有在自定义ViewGroup中才用到)   
绘制：onDraw 控制显示在屏幕上的样子(viewgroup没有这个过程)**

**这个过程一般是异步的，所以，有时候我们去获取view的宽高时，有可能以上三个步骤还没有完成，所以获取不到。所以我们要怎样在view绘制完成之前获取到view的宽高呢？**

1. **在代码中使用的方法：手动调用测量方法**

**//制定测量规则 参数表示size + mode   
int width =View.MeasureSpec.makeMeasureSpec(0,View.MeasureSpec.UNSPECIFIED);   
int height =View.MeasureSpec.makeMeasureSpec(0,View.MeasureSpec.UNSPECIFIED);   
view.measure(width,height);   
//调用measure方法之后就可以获取宽高。   
int height=view.getMeasuredHeight();   
int width=view.getMeasuredWidth();**

**2）设置addOnGlobalLayoutListener监听**

**ViewTreeObserver vto =view.getViewTreeObserver();   
vto.addOnGlobalLayoutListener(new OnGlobalLayoutListener(){   
@Override   
public void onGlobalLayout() {   
view.getViewTreeObserver().removeGlobalOnLayoutListener(this);   
int height =view.getMeasuredHeight();   
int width =view.getMeasuredWidth();   
}   
});**

**ViewTreeObserver可以添加很多监听，其中addOnGlobalLayoutListener是注册一个回调函数，当在一个视图树中全局布局发生改变或者视图树中的某个视图的可视状态发生改变时调用这个回调函数。但是需要注意的是OnGlobalLayoutListener可能会被多次触发，因此在得到了高度之后，要将OnGlobalLayoutListener注销掉。**

**3）增加addOnPreDrawListener监听**

**ViewTreeObserver vto =view.getViewTreeObserver();**

**vto.addOnPreDrawListener(new ViewTreeObserver.OnPreDrawListener() {   
@Override   
public boolean onPreDraw() {   
int height =view.getMeasuredHeight();   
int width =view.getMeasuredWidth();   
}   
});**

1. **LayoutParams的总结**

**LayoutParams这个类，其实一直在用，但是没有去做过一个很好的总结，最近的开发过程中，经常需要再代码里优化UI，所以LayoutParams用得会比较多，所以总结一下。**

**LayoutParams只是ViewGroup的一个内部类，所以ViewGroup里边这个LayoutParams类是 baseClass 基类，实际上每个不同的ViewGroup都有自己的LayoutParams子类。比如LinearLayout和RelativeLayout。这个LayoutParams类是用于child view（子视图） 向 parent view（父视图）传达自己的意愿的一个东西（孩子想变成什么样向其父亲说明）LayoutParams主要设置的是子view在父类布局的参数。**

**所以XML文件中的父布局的android属性其实和在代码中通过LayoutParams设置的属性是同一个概念。**

**一般通过view.getLayoutParams() 获取到的都是父布局对应的LayoutParams。**

**所有ViewGroup的子类，都是可以通过LayoutParams来设置起布局的，而我们平时用到的大部分布局都是ViewGroup的子类，比如LinearLayout（RadioGroup，TableLayout，TabWidget等），RelativeLayout（DialerFilter，TwoLineListItem等），FrameLayout（ScrollView，ImageSwitcher，ViewFlipper等），AdapterView（GridView，RecyclerView，ListView等）。所有的这些ViewGroup在使用或者自定义的过程中，都可以通过LayoutParams来对布局做不同的修改，以满足我们的需求。**

**MarginLayoutParams是继承自ViewGroup.LayoutParmas,是VideoGroup的一个内部类，一般VidwGroup 的LayoutParmas都是直接继承MarginLayoutParams，所以可以直接通过LayoutParmas来设置或者获取margin值，但ViewGroup的LayoutParmas不能直接获取marge值，需要先获取MarginLayoutParams,再通过MarginLayoutParams可以获取或者修改view的margin值。**

**https://blog.csdn.net/liuhaomatou/article/details/22899925/**

1. **MVP使用总结**

**Video项目中使用了MVP的框架，以前开发没有严格的使用框架，没有什么概念，经常在博客上看到mvp，mvc，但是这样看博客，其实能感受到的东西比较少，只有在项目中使用过，才真的能发现框架还是有他的优越性。**

1. **MVP框架会使整个项目显示逻辑十分清楚，view层就是做UI更新的，view层持有一个Presenter类，整个view里面除一个Presenter类外，其他成员变量基本都是更新UI的相关类。这样，view层的逻辑就简单了，涉及到逻辑处理，则肯定需要调用Presenter去处理逻辑，因为view层本身是不会去处理的。View层也不会出现那么多杂乱和随意的内部方法，代码结构比较清楚。Presenter层会持有一个view对象和一个model对象，这就是沟通view和model的桥梁。当view层需要处理逻辑时，会通过view持有的Presenter来调用Presenter里面的方法，在Presenter里面，会去做那些复杂的逻辑运算，当需要数据绑定的时候，Presenter会通过持有的model对象去获取数据，最好处理完逻辑之后，大部分情况是会通过UI更新来体现给用户的，这时就通过Presenter持有的view对象去做UI的更新即可。Model层主要是去做获取数据的工作，他只会被Presenter持有，view层不直接与Model层交互。**
2. **View，Model，Presenter层一般都会分别实现一个对应的接口，用来把当前层的一些方法抽象出来。这样做利于代码架构的清晰，当我们拿到需求时，我们直接考虑页面需要干什么，把view层的方法先抽象出来成为view的接口，通过页面需求，再得出Presenter层需要做的逻辑处理，同样出现出来成为Presenter的接口，同样整个页面涉及到什么样的数据更新，抽象出来成为Model的接口，这样，我们开发过程的最开始，就已经有整个需求的框架逻辑了，思路也会比较清晰。另外，对于别人而言，只需要看这三个接口，就大概了解这个页面做了什么事。会使代码逻辑比较简洁明了。**
3. **但是mvp也还是有一些缺点的，也可能是自己理解的不够深的误读，使用mvp模式时，代码的逻辑性变得比较强，但是，他要求我们必须要遵循mvp的要求，经常出现一个简单的ui更新，需要通过Presenter去处理一个超简单的逻辑，然后再回调回来进行ui更新，其实这个过程就有点矫枉过正了，但是如果在view层去直接更新，一是有点违背mvp的原理，再是可以需要去获取一些本不该在view层出现的逻辑处理的实体类。但是总体而言，还是感觉mvp使代码逻辑非常清晰，扩展性也比较好。**
4. **CoordinatorLayout布局的使用**

**由于开发需要的需要，对CoordinatorLayout来实现标题页的一些特殊效果，所以对CoordinatorLayout做了一些了解，里面踩了很多的坑，具体的分析在另一篇总结中，就不做重复的总结。**

1. **自定义view的相关总结**

**版本1.0开发中，我做的页面部分比较多，所以对整体UI调整还有自定义view的使用比较多，稍微总结一下自定义view的使用感受。**

**一个项目中，由于整体UI风格比较像，所以，我们在自定义view的时候，尽量考虑view的扩展，如果是经常使用的view，一些可变的参数值最好不要写死，通过一些设置参数，在使用的时候去进行设置，可以做到更灵活。**

**总结使用过的自定义view的种类，做了以下总结：**

1. **继承与一些原生控件（比如imageView, textView, RecyclerView等）的自定义view**

**这种自定义view会保留原有控件的属性，一般是对控件做一些个性化处理以满足需求。**

1. **继承ViewGroup(包括ViewGroup，XXXLayout等)**

**这种自定义view相对来说是我们用到最多的，其实就是把基本控件组合起来成一个新控件，相当于将一个XML写成一个View来使用。这种view又有几种细分：**

1. **简单的基本控件组合，这种就是简单的把一个XML用一个view表示出来，同时，添加一些view的各种设置和处理模块。**
2. **非简单控件的组合，需要在整个view或者一些子view绘制的时候，测量view的大小，确定view 的位置，绘制一些额外的东西，这其实就是对应到onMeasure(), onLayout(), onDraw()。**

**有时需要还需要做一些不同的时间拦截和响应，需要在onInterceptTouchEvent() 和onTouchEvent()中去做处理。**

1. **自绘view，一般继承自view**

**这一类自定义控件是三类自定义view里面最复杂的，但也是最强大的。注重控件本身的特性，如形状、动画效果等，一般继承View。他本身的view就是不确定的，需要重写onMeasure(), onLayout()来测量和布局，重写onDraw()方法完成绘制。而且，有时候可以通过不断重绘这个view来实现动画效果，功能很强大。**