17个XML布局小技巧

yechaoa 鸿洋 2022-10-08 08:35 发表于安徽

本文作者

作者：**yechaoa**

链接：
https://juejin.cn/post/7145861715798802462
本文由作者授权发布。

整了3天弹框3了，还没搞定，欲哭无泪...还是照常上钟推文吧。

0 前言

我们开发时接触最多的就是xml布局了，还记得我们写Android的第一个Hello World吗，就是通过activity_main.xml显示出来的。

虽然xml写的很多，而且也没有什么技术难度，但是，这也往往是我们最容易忽略的地方，写xml不难，写出好的xml还是得下点功夫了。

什么算是好的xml布局呢，我认为核心有两点，一个是提升开发效率，另一个是提升app性能。围绕着这两点，我也精心整理出了17个xml布局小技巧，下面一起来看看都有哪些，你又掌握了几个呢？

1 Space

官网是这么介绍的：

Space 是一个轻量级的 View 子类，可用于在通用布局中创建组件之间的间距。

为什么说是轻量级呢，是因为Space的draw方法是空的，也就是什么都不绘制，只有onMeasure方法测量宽高。

来看下源码：

```
public final class Space extends View {

    /**
     * Draw nothing.
     *
     * @param canvas an unused parameter.
     */
    @Override
    public void draw(Canvas canvas) {
    }

    //...

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        setMeasuredDimension(
                getDefaultSize2(getSuggestedMinimumWidth(), widthMeasureSpec),
                getDefaultSize2(getSuggestedMinimumHeight(), heightMeasureSpec));
    }
}
```

所以Space作用于组件之间的间距时，绘制效率更高，特别是在需要动态修改间距时，这点尤为体现。

比如你要动态修改组件的margin，如果用Space来当间距，只需要修改Space的宽度或高度即可，因为减少了绘制流程，所以比重绘其他组件更高效。

使用起来也很简单：

```
<Space
    android:id="@+id/space"
    android:layout_width="20dp"
    android:layout_height="20dp"/>
```

如果你想，Space完全可以替代margin，但是不一定能替代padding，因为padding是内边距，假如padding有背景色的话，就不能用Space代替了，因为Space的draw方法什么都不绘制的原因，所以也不会有背景色，除非背景色是在父view里设置的。

2 GuideLine

ConstraintLayout自2018年发布第一个正式版本以来，已经4年多了，它通过扁平化的布局方式，有效的解决了层级嵌套的问题，不仅比RelativeLayout更灵活，而且性能上更佳，再配合上可视化工具拖拽编辑，效率上也有大大的提升，如果你还没有用上，建议你一定要尝试一下。

而在使用ConstraintLayout的过程中，我发现有些同学总是会忽略GuideLine，尽管ConstraintLayout已经非常好用了，但是有些布局仍然显得有些「笨拙」。而如果你能妙用GuideLine，你会发现，布局越来越简单，适配也越来越方便。

GuideLine是ConstraintLayout布局的辅助对象，仅用于布局定位使用，它被标记了View.GONE，并不会显示在设备上。

来看下源码：

```
public class Guideline extends View {
  public Guideline(Context context) {
    super(context);
    super.setVisibility(View.GONE);
  }

  public Guideline(Context context, AttributeSet attrs) {
    super(context, attrs);
    super.setVisibility(View.GONE);
  }

  public Guideline(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    super.setVisibility(View.GONE);
  }

  public Guideline(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
    super(context, attrs, defStyleAttr);
    super.setVisibility(View.GONE);
  }

  //...
  @SuppressLint("MissingSuperCall")
  @Override
  public void draw(Canvas canvas) {

  }

  @Override
  protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(0, 0);
  }
  //...
}
```

标记为View.GONE是这句super.setVisibility(View.GONE)设置的默认值，不显示还是因为draw方法为空，跟上面的Space同出一辙。

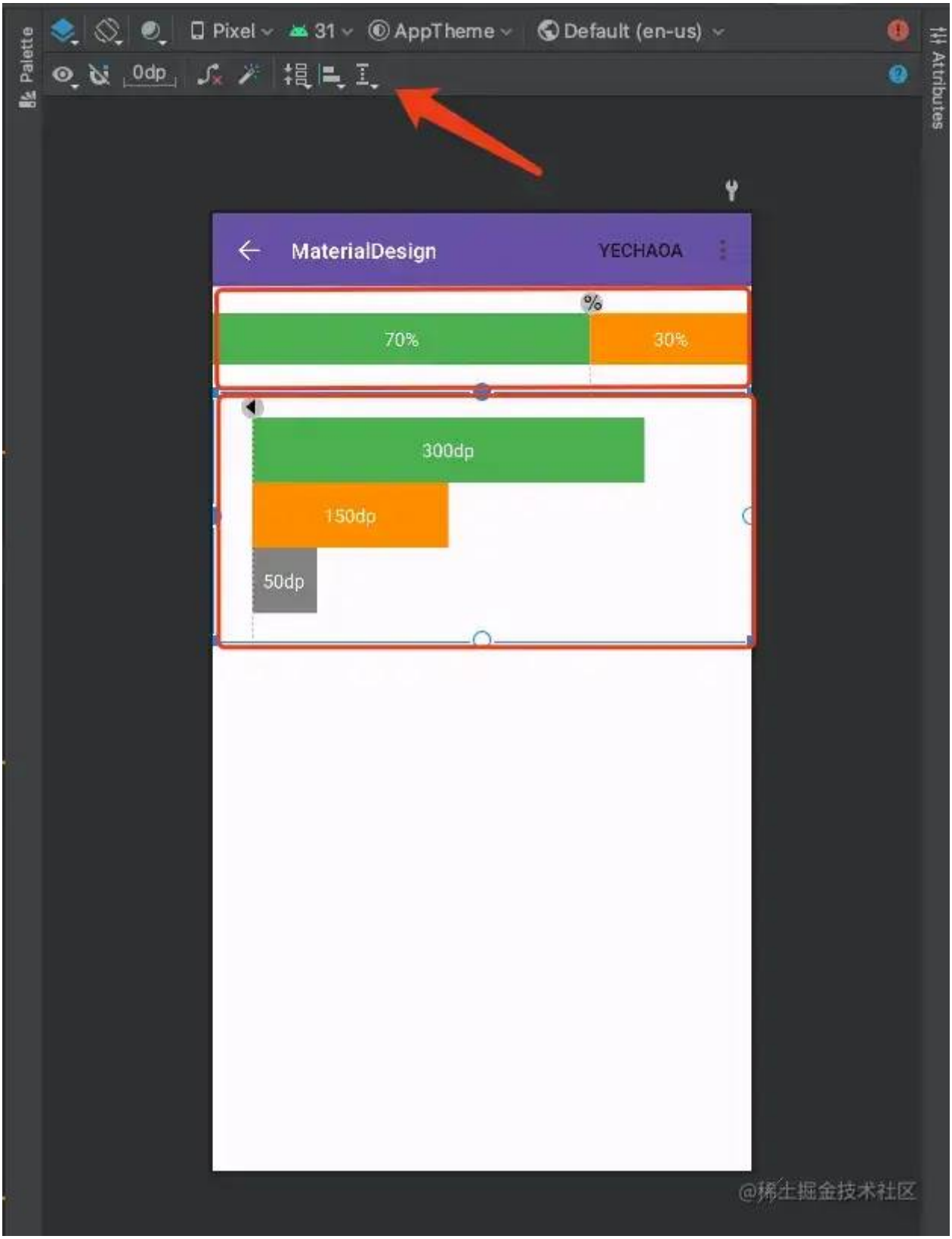
GuideLine可以通过3种不同的方式来辅助定位：

- layout_constraintGuide_begin 指定距布局左侧或顶部的固定距离
- layout_constraintGuide_end 指定距布局右侧或底部的固定距离
- layout_constraintGuide_percent 指定布局宽度或高度的百分比

同时也可以指定不同的方向：

- horizontal 垂直参考线
- vertical 水平参考线

下面简单演示一下效果：



1. 箭头所指处即创建GuideLine的地方，当然也不止GuideLine，比如还有Barrier
2. 第一个红框里是水平参考线，70%定位，用百分比能很好的解决适配问题，而我们常规的做法是使用LinearLayout嵌套然后设置子view的weight，虽然嵌套一层不多，但那也是嵌套，就像怀孕一样，你不能说只怀了一点点...

3. 第二个红框里是垂直参考线，距离左边30dp，这种情况适合多个子view向一个目标距离对齐，同样减少了层级嵌套问题，省得再嵌套一层设置padding，或者多个子view分别设置margin。而右边如果想要指定一个位置换行，可以了解一下Barrier~

xml代码就不贴了，已上传到Github：
https://github.com/yechaoa/MaterialDesign

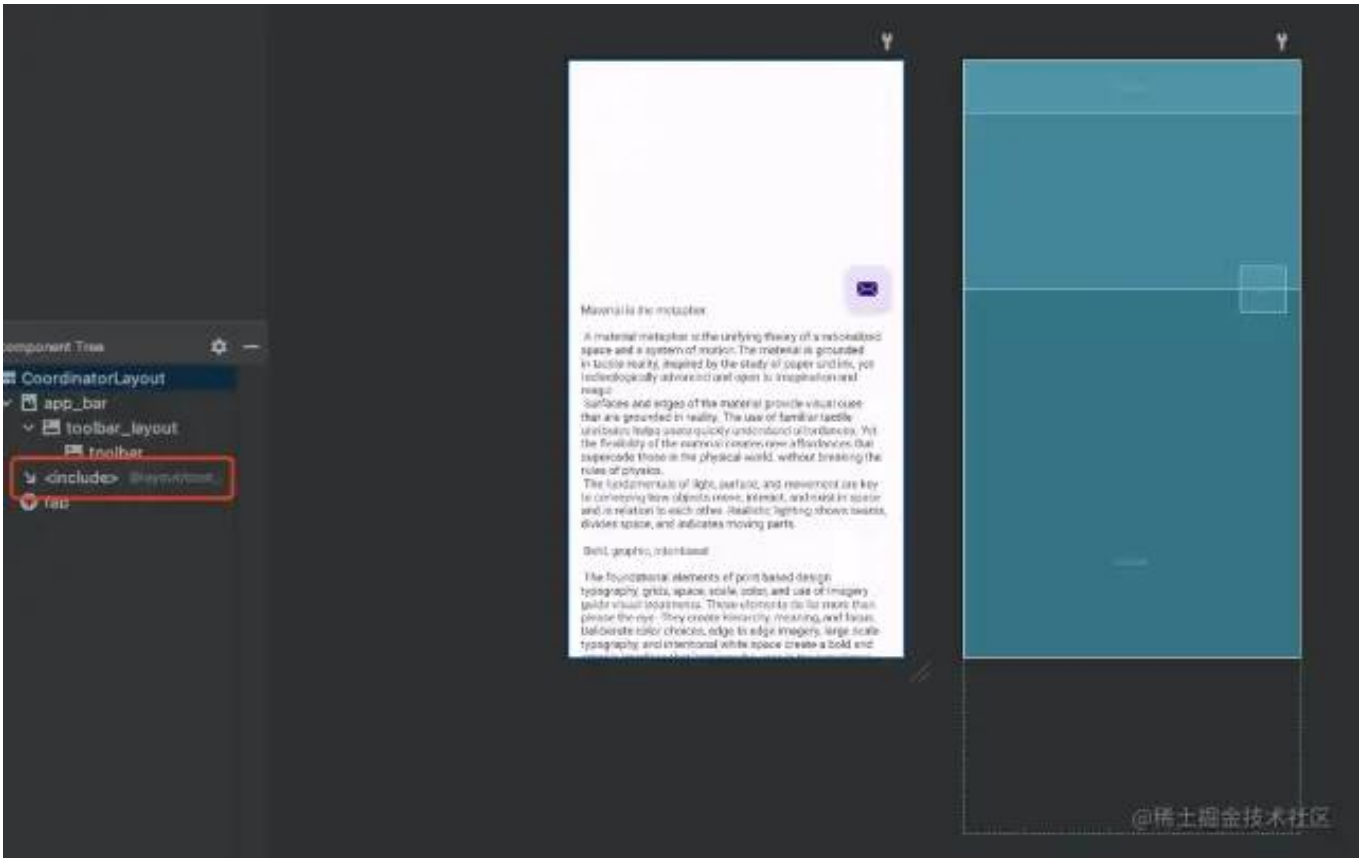
3 include

当我们在写一个复杂的页面时，xml代码可能有几百行甚至几千行，阅读起来总是很麻烦，如果又有很多的RelativeLayout嵌套的话，各个组件之间依赖关系错综复杂，看起来更是头大。

这时候就可以考虑抽取一波，用总分总的模式分为header、content、footer，进一步把内容区抽成一个一个的独立的子layout，然后使用include标签把它们分别引进根布局，这就跟我们项目架构设计一个意思，一个壳工程加n个子模块。

子layout只需要负责处理好自己内部的布局，统筹交给父layout，这样总体就比较清晰，想了解细节再去看子layout即可。

比如：



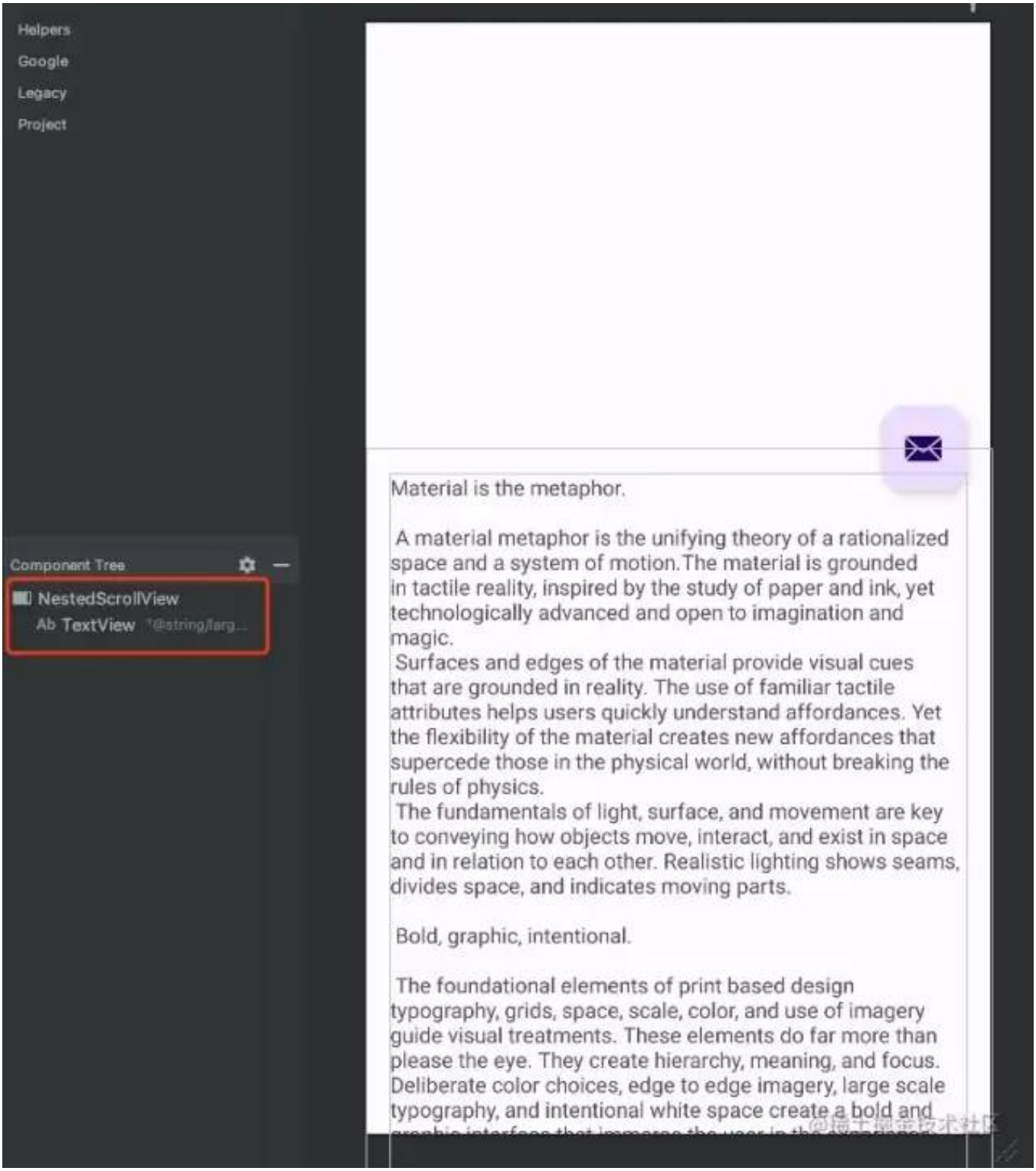
```
<include layout="@layout/content_scrolling"/>
```

content_scrolling即是我们抽出去的子layout。

4 tools:showIn

这个属性一般是配合include标签使用的。当我们把子layout抽出去之后，它的布局是相对独立的效果，但是总归要include到根布局的，如果能在子layout布局的时候看到它在父layout里面的效果，那就事半功倍了。

上面的content_scrolling.xml：



实际上布局只有一个TextView，但是在预览视图中还可以看到FloatingActionButton，这就是使用了tools:showIn属性，当子layout嵌入在父layout中时，只需要使用tools:showIn在子layout的根布局指定父layout，就可以实时预览在父layout中的效果了。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.core.widget.NestedScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:layout_behavior="@string/appbar_scrolling_view_behavior"
  tools:context="com.yechaoa.materialdesign.activity.CollapsingToolbarActivity"
```

```
    tools:showIn="@layout/activity_collapsing_toolbar">

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_margin="@dimen/text_margin"
      android:text="@string/large_text"/>

</androidx.core.widget.NestedScrollView>
```

即：tools:showIn="@layout/activity_collapsing_toolbar"。

5 ViewStub

ViewStub是一个轻量级的条件视图组件。在做类似页面秒开这类性能优化时，是比较常见的延迟加载手段。

轻量级是因为ViewStub跟Space一样draw方法为空。

条件视图的场景比如，当我们需要根据条件判断来显示哪个view的时候，一般都会把每个场景的view都写在页面中，然后根据条件分别设置view的visibility，这样做的缺点是，即使view是View.GONE，但是在页面渲染加载的时候仍会实例化创建对象，并初始化它的属性，很明显这是浪费资源的，所以这个时候用ViewStub是一个很好的优化手段。

当我们明确知道需要显示哪个view的时候，通过ViewStub把实际视图inflate进来，这样就避免了资源浪费。

只有调用了ViewStub.inflate()的时候布局才会加载，才会创建对象实例化。

示例：

```
<ViewStub
  android:id="@+id/stub_import"
  android:inflatedId="@+id/panel_import"
  android:layout="@layout/progress_overlay"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom" />
```

inflate：

```
findViewById<View>(R.id.stub_import).visibility = View.VISIBLE
```

```
// or
val importPanel: View = findViewById<ViewStub>(R.id.stub_import).inflate()
```

6 tools:text

TextView是我们使用的最多的一个组件了，经常有这样的需求，“标题显示不下用...代替”，是不是很熟悉。

如果标题是一个动态数据，默认显示app name，拿到数据后再更新。这种情况，一般都是在android:text里面加字符来调试，调试完了再改成默认的app name，其实也不用这么麻烦，直接默认app name，然后使用tools:text属性就可以预览字符超限的效果。

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/item_textView"
  android:layout_width="100dp"
  android:layout_height="wrap_content"
  android:maxLines="1"
  android:ellipsize="end"
  android:text="TextView"
  tools:text="TextViewTextView" />
```

默认文案还是用android:text显示，超限的效果用tools:text预览即可，实际效果还是android:text，tools:text只是方便我们调试预览，提高效率，减少编译等待时间。

7 tools:visibility

这个属性是用来预览不显示的View。

比如在“个人中心”页面需要在昵称后面给个文案提示“开通会员”，默认不显示，即android:visibility="gone"，判断不是会员后才显示文案，但是在开发的过程中需要调试会员和非会员的两种显示效果，即可以通过tools:visibility="visible"来预览显示的效果，省得再编译运行造数据了，方便又提效。

代码示例：

```
<TextView
   tools:visibility="visible"
   android:visibility="gone"
   android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:background="@color/greenPrimary"
    android:gravity="center"
    android:padding="10dp"
    android:text="开通会员"
    android:textColor="@color/white" />
```

8 RecyclerView相关6个

RecyclerView也是我们使用非常高频的一个组件了，一般会在xml中这么定义RecyclerView：

```
<androidx.recyclerview.widget.RecyclerView
 android:id="@+id/recycleView"
 android:layout_width="match_parent"
 android:layout_height="wrap_content" />
```

效果是这样的：

这样其实完全看不出RecyclerView在页面中显示的效果，只能每次编译运行看效果，而每次编译运行无疑会花费我们很多宝贵的时间，下面就介绍几个可以帮助大家提效的属性。

tools:listitem

我们可以通过设置tools:listitem属性来预览item的显示效果，tools:listitem属性指定的是一个layout

```
tools:listitem="@layout/item_main"
```

效果：

tools:itemCount

预览item在RecyclerView中显示设置数量的效果，比如：

```
tools:itemCount="3"
```

即会显示3个item的效果。

tools:listheader

```
tools:listheader="@layout/item_header"
```

效果同tools:listitem

tools:listfooter

效果同tools:listitem

```
tools:listfooter="@layout/item_footer"
```

app:layoutManager

上面RecyclerView的效果是默认垂直方向的，我们都知道RecyclerView必须要设置一个layoutManager才可以显示出来，我们通常会用代码来设置，比如：

```
mBinding.recycleView.layoutManager = GridLayoutManager(this, 2)
```

实际上layoutManager也是可以在xml中通过app:layoutManager属性来设置的，比如：

```
app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
```

默认的LinearLayoutManager是垂直方向的，如果我们想要改方向可以通过android:orientation属性，比如：

```
android:orientation="horizontal"
```

这样就可以在编写xml的时候顺手就加上了，既可以查看预览效果，也避免了代码忘记设置的尴尬情况。

app:spanCount

上面的示例中RecyclerView的layoutManager指定了LinearLayoutManager，我们还可以指定为GridLayoutManager，但是GridLayoutManager默认的spanCount是1，如果我们需要设置spanCount为2，那该怎么预览呢，这时候就用到了app:spanCount属性，可以指定需要显示的列数。

```
app:spanCount="2"
```

效果：

9 android:tint

着色器，这个属性在之前的包体积优化中有提到，可以减少图片数量，从而减小包大小。

我们通常会用ImageView显示一张图片，比如原本是一个白色的返回icon，现在另一个地方要用黑色的了，就不需要使用黑白两张图了，而是使用tint来修改为黑色即可，当然，也有局限，适合纯色图片。

效果：

示例：

```
<LinearLayout
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="30dp"
  android:orientation="horizontal">

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@color/black"
    android:src="@mipmap/ic_back" />

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@color/red"
    android:src="@mipmap/ic_back"
    app:tint="@color/black" />

</LinearLayout>
```

在appcompat的高版本中已经改用app:tint代替。

代码方式修改tint：

```
mBinding.imageView.imageTintList = ContextCompat.getColorStateList(this, R.color.greenPrimary)
```

除了tint还有backgroundTint，效果同理。

使用场景除了上面的示例外，还可以在点赞、收藏这类场景的显示上使用。

10 android:divider

LinearLayout也是我们使用非常高频的一个Layout，下面介绍两个个少为人知的属性。
相信很多人都用View写过分割线的效果，类似这样：

```
<TextView />

<View
```

```
    android:layout_width="match_parent"
    android:layout_height="1dp"
    android:background="#EEEEEE" />

  <TextView />
```

如上，当有多个TextView之间需要添加分割线的时候，就只能一个一个复制，复制其实也没什么，就是代码看起来不优雅。

其实有个比较优雅的办法，LinearLayout可以通过android:divider属性添加分割线，结合android:showDividers属性即可达到效果。

xml：

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="20dp"
    android:background="@drawable/shape_radius5_white"
    android:divider="@drawable/shape_divider_linear"
    android:orientation="vertical"
    android:showDividers="middle" >

    <TextView
        style="@style/MyTextView"
        android:text="删除个人信息"
        app:drawableStartCompat="@mipmap/ic_helper" />

    <TextView
        style="@style/MyTextView"
        android:text="注销账户"
        app:drawableStartCompat="@mipmap/ic_helper" />

    <TextView
        android:id="@+id/tv_about"
        style="@style/MyTextView"
        android:text="关于我们"
        app:drawableStartCompat="@mipmap/ic_helper" />

</LinearLayout>
```

shape_divider_linear是分割线的样式：

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:left="50dp" >
        <shape android:shape="rectangle">
            <solid android:color="#F6F6F6" />
            <size android:height="1dp" />
        </shape>
    </item>
</layer-list>
```

效果：

showDividers有4个选项：

- middle 每两个组件间显示分隔线
- beginning 开始处显示分隔线
- end 结尾处显示分隔线
- none 不显示

其实举一反三，除了分割线，View之间的间隔也可以这么实现，省得每个子view都要写margin。

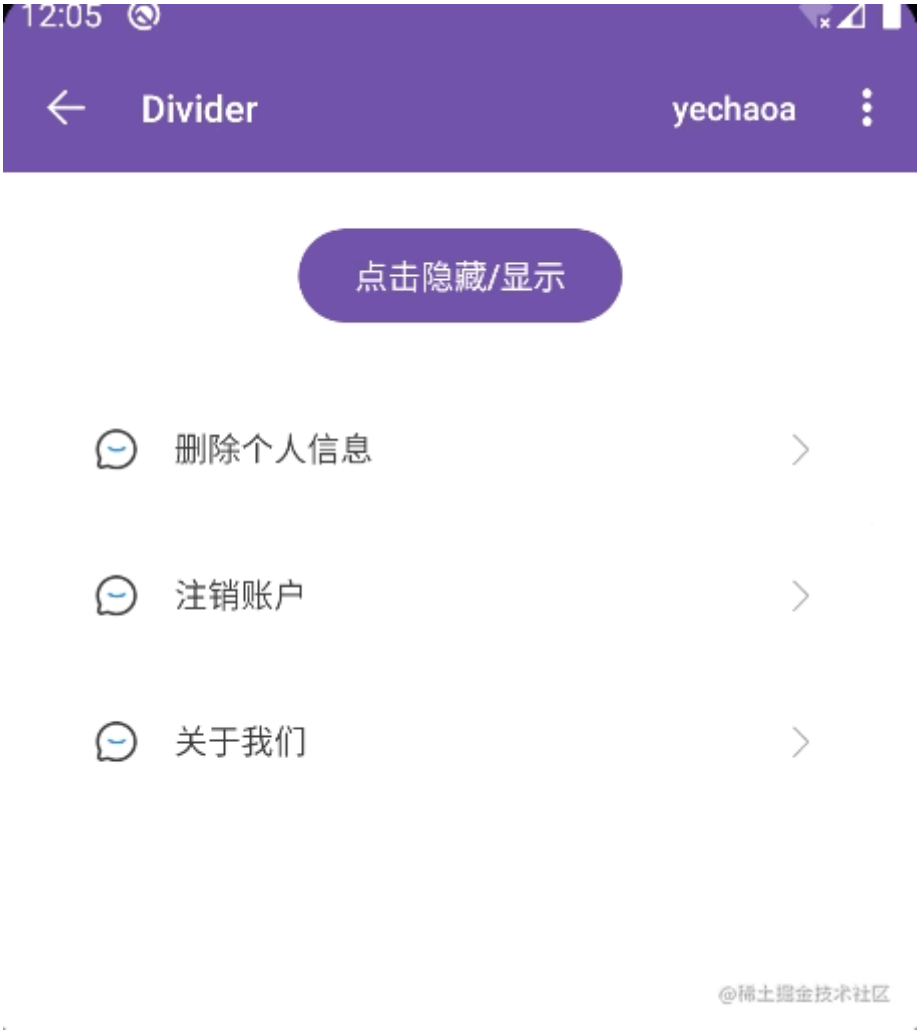
11 android:animateLayoutChanges

animateLayoutChanges属性是ViewGroup里面的，主要是在子view的添加和移除时，添加一个默认300ms的渐变动画。

代码：

```
android:animateLayoutChanges="true"
```

效果：



默认添加移除操作是比较生硬的，加上动画之后体验上会好很多。

当然，如果你想修改默认动画也是可以的。怎么修改？没有比学习源码更直接的了。

源码：

```
case R.styleable.ViewGroup_animateLayoutChanges:
  boolean animateLayoutChanges = a.getBoolean(attr, false);
  if (animateLayoutChanges) {
    setLayoutTransition(new LayoutTransition());
  }
  break;
```

当animateLayoutChanges属性值为true时，调用setLayoutTransition方法，并传入一个默认的LayoutTransition对象。

LayoutTransition对象用于构造动画，跟一般的动画使用差不多，感兴趣的可以看下官方文档或者跟下源码。
https://developer.android.google.cn/reference/android/animation/LayoutTransition?hl=en

自定义LayoutTransition对象之后，调用ViewGroup.setLayoutTransition(LayoutTransition)即可。

12 android:foreground

```
android:foreground="?android:attr/selectableItemBackground"
```

在Android5.0以后，给View加上这个属性之后，点击时默认会有一个水波纹的效果，一般可点击的View默认都有这个效果，比如Button，一般通常会在自定义的item view上加上这个属性用来提升用户体验。

最后

如上，本文一共介绍了17个在日常编写xml的过程中对提升效率和提升性能的属性，如果你也有心得，欢迎评论补充。

如果本文对你有一丢丢帮助，也感谢点赞支持~
https://github.com/yechaoa/MaterialDesign

相关文档

1. Space

https://developer.android.google.cn/reference/kotlin/android/widget/Space?hl=en

2. Guideline

https://developer.android.google.cn/reference/androidx/constraintlayout/widget/Guideline

3. ConstraintLayout

https://developer.android.google.cn/training/constraint-layout

4. ViewSub

https://developer.android.com/training/improving-layouts/loading-ondemand

5. tools

https://developer.android.google.cn/studio/write/tool-attributes?hl=en

6. LayoutTransition

https://developer.android.google.cn/reference/android/animation/LayoutTransition?hl=en

最后推荐一下我做的网站，玩Android: *wanandroid.com* ，包含详尽的知识体系、好用的工具，还有本公众号文章合集，欢迎体验和收藏！

推荐阅读：

Android人涅槃重生之路：更新+1

Jetpack 中被“雪藏”的状态保存利器！

Google 为何把 SurfaceView 设计的这么难用？