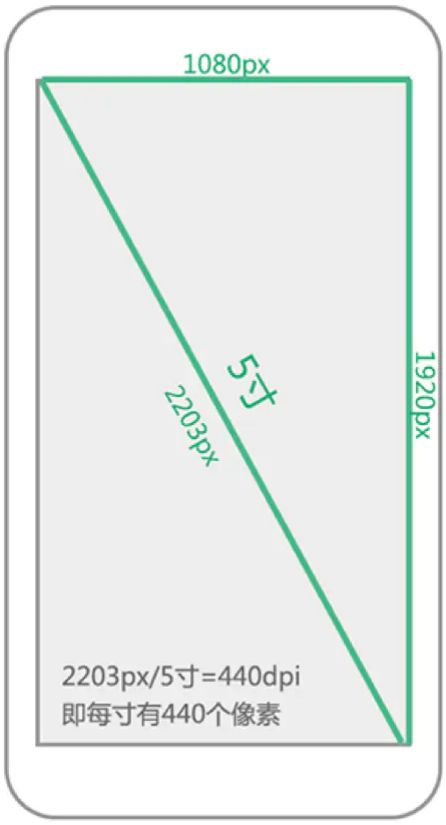
屏幕适配

Android运行在各种各样的设备上，他们有着不同的屏幕尺寸与分辨率。为了界面能够更加美观地呈现，需要开发者做出进一步的优化。

屏幕兼容性概览

术语



图片来自 Android 屏幕适配：最全面的解决方案

屏幕尺寸

屏幕尺寸指手机屏幕对角线的物理尺寸，单位英寸（inch），目前常见的屏幕尺寸为5.5~7英寸。

```
1英寸 = 2.54 厘米
```

像素

像素（px），作为屏幕显示的最小单位存在。

屏幕分辨率

屏幕分辨率指的是屏幕上的总像素数。通常使用纵向像素点数*横向像素点数表示。例如1920*1080

屏幕像素密度

屏幕像素密度指的是单位英寸上的像素点数。单位为dpi（dots per inch），也称为ppi（pixels per inch）。一般情况下，dpi用于印刷行业，表示每英寸打的点数；ppi更适合用于电子显示，表示每英寸的像素点数。

dpi是通过build.prop属性文件进行配置的，开发人员不能进行修改。获取build.prop配置信息最简便的方式就是通过ADB命令：

> adb shell getprop（这个命令可以获取所有的配置信息）

如果需要获取其中的指定信息，可以在后面加上需要获取的配置项id，如：

> 获取像素密度
> adb shell wm density

```
L:\android\demo>adb -s ae60aa5a shell wm density
Physical density: 440
```

所以，当你通过分辨率计算像素密度得到的值与实际结果不符，这是正常的。因为Android中不是通过计算得到dpi的值的，而是通过读取配置文件的方式得到的。

密度无关像素

密度无关像素（dip、dp-- density independent pixel），即在不同的dpi的display中，每一个dp对应不定量的px。例如，在160dpi的显示中，1dp = 1px，但是在240dpi的显示中，1dp=1.5px。**这也是官方推荐使用dp而不是px的原因，因为dp可以根据dpi动态适配，而px是底层单位，1px就是1px，在不同分辨率的显示屏中，显示效果肯定是不适配的。**

dpi和dip都是密度，其中，dpi称为像素密度，表示的是单位inch中的像素数量；dip称为与密度无关的像素，表示的是不同的像素密度屏幕中，组成一个dp的像素数量。

ldpi、mdpi、hdpi、xhdpi、xxhdpi、xxxhdpi、nodpi、tvdpi

密度限定符，表示不同的密度级别。不同的密度级别取值见android.util.DisplayMetrics

密度限定符	说明
`ldpi`	适用于低密度 (ldpi) 屏幕 (~ 120dpi) 的资源。
`mdpi`	适用于中密度 (mdpi) 屏幕 (~ 160dpi) 的资源（这是基准密度）。
`hdpi`	适用于高密度 (hdpi) 屏幕 (~ 240dpi) 的资源。
`xhdpi`	适用于加高 (xhdpi) 密度屏幕 (~ 320dpi) 的资源。
`xxhdpi`	适用于超超高密度 (xxhdpi) 屏幕 (~ 480dpi) 的资源。
`xxxhdpi`	适用于超超超高密度 (xxxhdpi) 屏幕 (~ 640dpi) 的资源。
`nodpi`	适用于所有密度的资源。这些是与密度无关的资源。无论当前屏幕的密度是多少，

	系统都不会缩放以此限定符标记的资源。
`tvdpi`	适用于密度介于 mdpi 和 hdpi 之间的屏幕（约 213dpi）的资源。这不属于“主要”密度组。它主要用于电视，而大多数应用都不需要它。对于大多数应用而言，提供 mdpi 和 hdpi 资源便已足够，系统将视情况对其进行缩放。如果您发现有必要提供 tvdpi 资源，应按一个系数来确定其大小，即 1.33*mdpi。例如，如果某张图片在 mdpi 屏幕上的大小为 100px x 100px，那么它在 tvdpi 屏幕上的大小应该为 133px x 133px。

为什么需要在不同的dpi下放入相对应的备用资源呢？主要还是为了更好的适配当前设备。例如：在加载drawable资源的时候，由于系统在对应的dpi文件夹找不到合适的资源，就会从其他dpi目录下查找，找到之后对其进行缩放，以适配当前的dpi设备。而这个缩放会引起图像的失真，故而需要在不同的目录下放置合适的图片。

在读取dpi限定符限定的资源时，系统的查找规则是偏向于向下查找最合适的资源。**例如：如果设备的屏幕密度为hdpi，当找不到确切资源时，应用会试着向上查找(xhdpi、xxhdpi)，若仍匹配不到资源，则从设备屏幕密度向下查找（从mdpi开始，其次是默认资源文件、ldpi）.**当找到最接近的资源时，系统会对其进行一个缩放操作，以保证不同dpi的资源在别的dpi设备下能够适配，缩放的操作就有可能是资源图片失真。例如：如果您的应用仅提供适用于中密度 (mdpi) 屏幕的位图，则在高密度屏幕上 Android 会将其放大，这样该图像在两种屏幕上会占用相同的物理空间。

图片资源自动适配导致失真的解决方案是使用SVG图片。SVG图形使用几何线条路径（而非像素）来定义插图，因此，它们可在不产生缩放失真问题的情况下绘制成任意尺寸。SVG图片的具体使用方式见这里。

具体参考Android资源图片读取机制

SP

即scale-independent pixels，可以根据文字大小首选项进行放缩，是设置字体大小的单位

android.util.DisplayMetrics

Android用于记录显示（display）信息的结构体，例如density，size以及font scaling.

如果需要访问DisplayMetrics的信息，需要做必要的初始化：

```
DisplayMetrics metrics = new DisplayMetrics();
// 当前display的信息记录在metrics中，可以通过metrics获取
getWindowManager().getDefaultDisplay().getMetrics(metrics);
```

DisplayMetrics记录了许多有用的信息，包括以下不同程度密度屏幕对应的量化DPI值。

```
    /**
     * low-density屏幕的标准DPI，即每英寸120个像素
     */
    public static final int DENSITY_LOW = 120;

    /**
     * medium-density屏幕的标准DPI，即每英寸160个像素
     */
    public static final int DENSITY_MEDIUM = 160;


    /**
     * high-density屏幕的标准DPI，即每英寸240个像素
     */
    public static final int DENSITY_HIGH = 240;


    /**
     * extra-high-density屏幕的标准DPI，即每英寸320个像素
     */
    public static final int DENSITY_XHIGH = 320;

    /**
     * extra-extra-high-density屏幕的标准DPI，即每英寸480个像素
     */
    public static final int DENSITY_XXHIGH = 480;

    /**
     * extra-extra-extra-high-density屏幕的标准DPI，即每英寸640个像素。一般来说，Android应用使用DENSITY_XHIGH就已经可以了。 对于xxxhdpi这种密度，使用场景一般为4K的电视屏幕-3840x2160，
     * 是传统高清屏的两倍，一般而言，传统高清屏为1920x1080，为xhdpi
     */
```

```
32      public static final int DENSITY_XXXHIGH = 640;
33
34      /**
35       * 系统使用的基准dpi，故而ldpi:mdpi:hdpi:xhdpi:xxhdpi:xxxhdpi =
 0.75:1:1.5:2:3:4，也就是说，当以mdpi设备为基准切一张30*30的图，放在hdpi
设备中会被缩小1.5倍，为了适配hdpi密度设备，应该在hdpi文件夹中放入一张45*45的
图片，即1:(30*30) = 1.5:(x:x)，计算结果为x=45
36       */
37      public static final int DENSITY_DEFAULT = DENSITY_MEDIUM;
38
39      /**
40       * 缩放因子，即dpi转换为密度比例，也就是说，默认1dpi对应160pixel
41       * @hide
42       */
43      public static final float DENSITY_DEFAULT_SCALE = 1.0f / DEN
SITY_DEFAULT;
44
45      /**
46       * 设备的稳定密度
47       * <p>
48       * 这个值是个运行期获取的常量，不再反射当前的display density.想要获取
特定的display的当前密度，使用{@link #densityDpi}
49       * 这个值是定义在系统属性配置文件中的，所以可以通过修改该属性来修改设备的
像素密度，这也正解释了为什么有些Android主板商家的屏计算下来dpi应该是mdpi，
50       * 但实际获取到的确实hdpi或者xhdpi，就是因为他们修改了这个属性值。
51       */
52      public static final int DENSITY_DEVICE_STABLE = getDeviceDen
sity();
53
54      private static int getDeviceDensity() {
55          // qemu.sf.lcd_density can be used to override ro.sf.lcd
_density
56          // when running in the emulator, allowing for dynamic co
nfigurations.
57          // The reason for this is that ro.sf.lcd_density is writ
e-once and is
58          // set by the init process when it parses build.prop bef
ore anything else.
59          // 应用运行在模拟器时，qemu.sf.lcd_density可用于覆盖ro.sf.lcd_
density，允许动态配置。原因是ro.sf.lcd_density只写一次，
```

```
60          // 并由init进程在解析build.prop（在任何步骤之前）时进行设置，也就是最先设置这个值且不可在任何运行过程中进行更改。
61          return SystemProperties.getInt("qemu.sf.lcd_density",
62                  SystemProperties.getInt("ro.sf.lcd_density", DENSITY_DEFAULT));
63      }
64
65      /**
66       * The absolute width of the available display size in pixels.
67       * 可用屏幕的绝对宽度（包括状态栏），单位为像素
68       */
69      public int widthPixels;
70
71      /**
72       * The absolute height of the available display size in pixels.
73       * 可用屏幕的绝对高度（包括状态栏），单位为像素
74       */
75      public int heightPixels;
76
77      /**
78       * display表示逻辑像素密度。是一个dp单位缩放因子（一个dp相当于160dpi屏上的一个像素，也就就是       * 说，在一个160dpi(mdpi)的屏上，1dp=1px，在一个320dpi(xhdpi)的屏上，1dp=2px）
79       *
80       * 这个值并不完全遵循实际的屏幕大小（由xdpi和ydpi给出，而是用于根据显示dpi中的总体更改按步骤缩       * 放整个UI的大小。例如，240x320屏幕的密度为1，即使其宽度为1.8英寸、1.3英寸
81       * 等等。但是，如果屏幕分辨率增加到320x480，但屏幕尺寸保持在1.5英寸x2英寸，则密度将增加（可能       * 达到1.5英寸）。
82       *
83       * 因为这个值与dpi有关，dpi越高，单位density表示的px就越大。对于同一个尺寸而言，密度越大，表示       * dpi越大，也就是density越大
84       *
85       * @see #DENSITY_DEFAULT
86       */
87      public float density;
88
89      /**
```

```
90      * The screen density expressed as dots-per-inch.  May be ei
    ther
91      * {@link #DENSITY_LOW}, {@link #DENSITY_MEDIUM}, or {@link
     #DENSITY_HIGH}.
92      * 显示密度，即屏幕上每英寸的像素点数。可能是{@link #DENSITY_LOW},
     {@link #DENSITY_MEDIUM}, or {@link #DENSITY_HIGH}.
93      * 例如，有设备尺寸为为5.5英寸，分辨率为1920*1080，通过勾股定理计算：d
    ensityDpi = Math.sqrt(Math.pow(1920, 2) + Math.pow(1080, 2)) /
     5.5 = 400
94      */
95     public int densityDpi;
96
97     /**
98      * A scaling factor for fonts displayed on the display.  Thi
    s is the same
99      * as {@link #density}, except that it may be adjusted in sm
    aller
100     * increments at runtime based on a user preference for the
     font size.
101     * 显示器上显示字体的缩放因子。与{@link#density}相同，但它可以在运行时
    根据用户对字体大小的偏好以较小的增量进行调整（sp转换为px的转换因子）
102     */
103    public float scaledDensity;
104
105    /**
106     * x维度下屏幕真实的像素密度
107     */
108    public float xdpi;
109    /**
110     * y维度下屏幕真实的像素密度
111     */
112    public float ydpi;
113
```

android.util.TypedValue

动态类型数据值的容器。主要用于保存资源值的资源。

```
```

public static float applyDimension (int unit, float value, DisplayMetrics metrics)

将包含dimension的未打包复合数据值转换为浮点值

Parameters

unit	转换的单位。例如：COMPLEX_UNIT_DIP表示将dp值转换为对应的float数值
value	转换的值
metrics	记录当前display的信息的结构体

```
    public static float applyDimension(int unit, float value, DisplayMetrics metrics) {
        switch (unit) {
        case COMPLEX_UNIT_PX:
            return value;
        case COMPLEX_UNIT_DIP:
            return value * metrics.density;
        case COMPLEX_UNIT_SP:
            return value * metrics.scaledDensity;
        case COMPLEX_UNIT_PT:
            return value * metrics.xdpi * (1.0f/72);
        case COMPLEX_UNIT_IN:
            return value * metrics.xdpi;
        case COMPLEX_UNIT_MM:
            return value * metrics.xdpi * (1.0f/25.4f);
        }
        return 0;
    }
```

根据官方的转换方法，可以得到各单位之间的转换关系：

1. 单位为DP，1dp = metrics.density
2. 单位为PX，1px = 1px
3. 1dp = metrics.density * 1px

注意：COMPLEX_UNIT_PT（1/72英寸）、COMPLEX_UNIT_IN（英寸）、COMPLEX_UNIT_MM（毫米），这些都是真实的物理单位，单位换算时使用的是metrics.xdpi

每个View初始化的时候都会将尺寸单位换算为px，见构造方法对attrs的解析过程。

资源限定符

见官方资源限定符说明

Android 如何查找最佳匹配资源

见官方Android 如何查找最佳匹配资源

适配方案

官方方案

1. 控件单位：dp
2. 使用 ConstraintLayout
3. wrap_content、match_parent、minHeight、minWidth、maxHeight、maxWidth、weight等属性
4. 使用.9和SVG
5. 创建备用布局
    - 使用最小宽度限定符

注意：该限定符是不区分方向的，min(w，h)的结果作为最小宽度

```
res/layout/main_activity.xml           # For handsets (smaller tha
n 600dp available width)
res/layout-sw600dp/main_activity.xml   # For 7” tablets (600dp wid
e and bigger)

当屏幕宽度大于等于600dp，则使用layout-sw600dp中的资源；否则使用layout中的资
源。
```

    - ·使用可用宽度限定符

最小宽度限定符是不区分方向的。有些情况下，你希望在屏幕宽度至少为600dp的情况下使用此布局，在屏幕宽度不足600dp（比如屏幕方向旋转）使用默认布局，这时候就可以使用可用宽度限定符。

```
    res/layout/main_activity.xml         # For handsets (smaller t
han 600dp available width)
    res/layout-w600dp/main_activity.xml  # For 7” tablets or any s
creen with 600dp
                                         #   available width (poss
ibly landscape handsets)
```

    - 屏幕方向限定符

如果希望当用户在纵向与横向之间切换屏幕方向时改变用户体验，可以将 `port` 或 `land` 限定符添加到资源目录名称中。

```
 res/layout/main_activity.xml                # For handsets
 res/layout-land/main_activity.xml           # For handsets in landscape
 res/layout-sw600dp/main_activity.xml        # For 7” tablets
 res/layout-sw600dp-land/main_activity.xml   # For 7” tablets in landscape
```

一 相同dpi不同分辨率

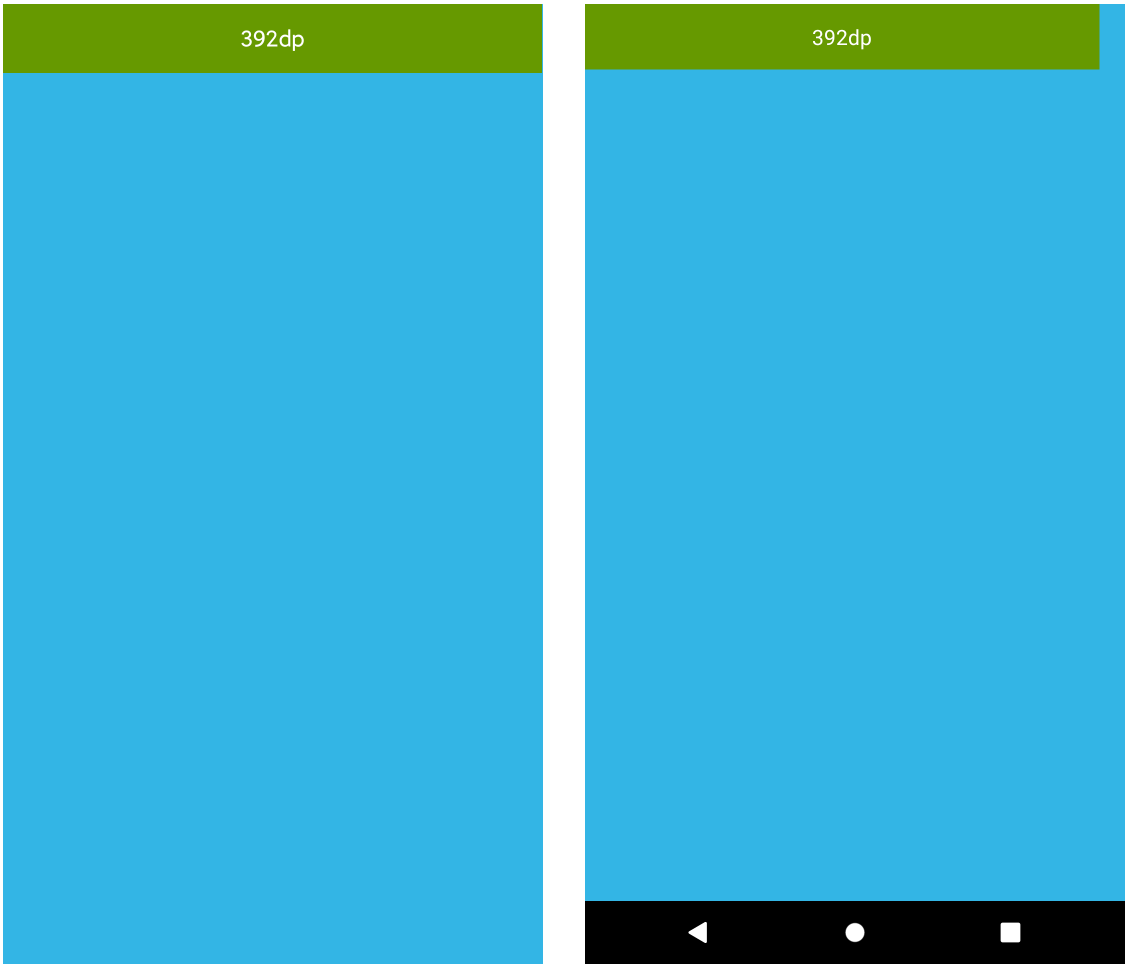
	分辨率	density	densityDpi	scaledDensity	xdpi	ydpi	物理尺寸
Pixel XL	1440 x 2560	3.5	560	3.5	560	560	5.5
Pixel 2XL	1440 x 2880	3.5	560	3.5	560	560	5.99

> 2560/3.5=731dp
> 2880/3.5=823dp

二 相同分辨率不同dpi

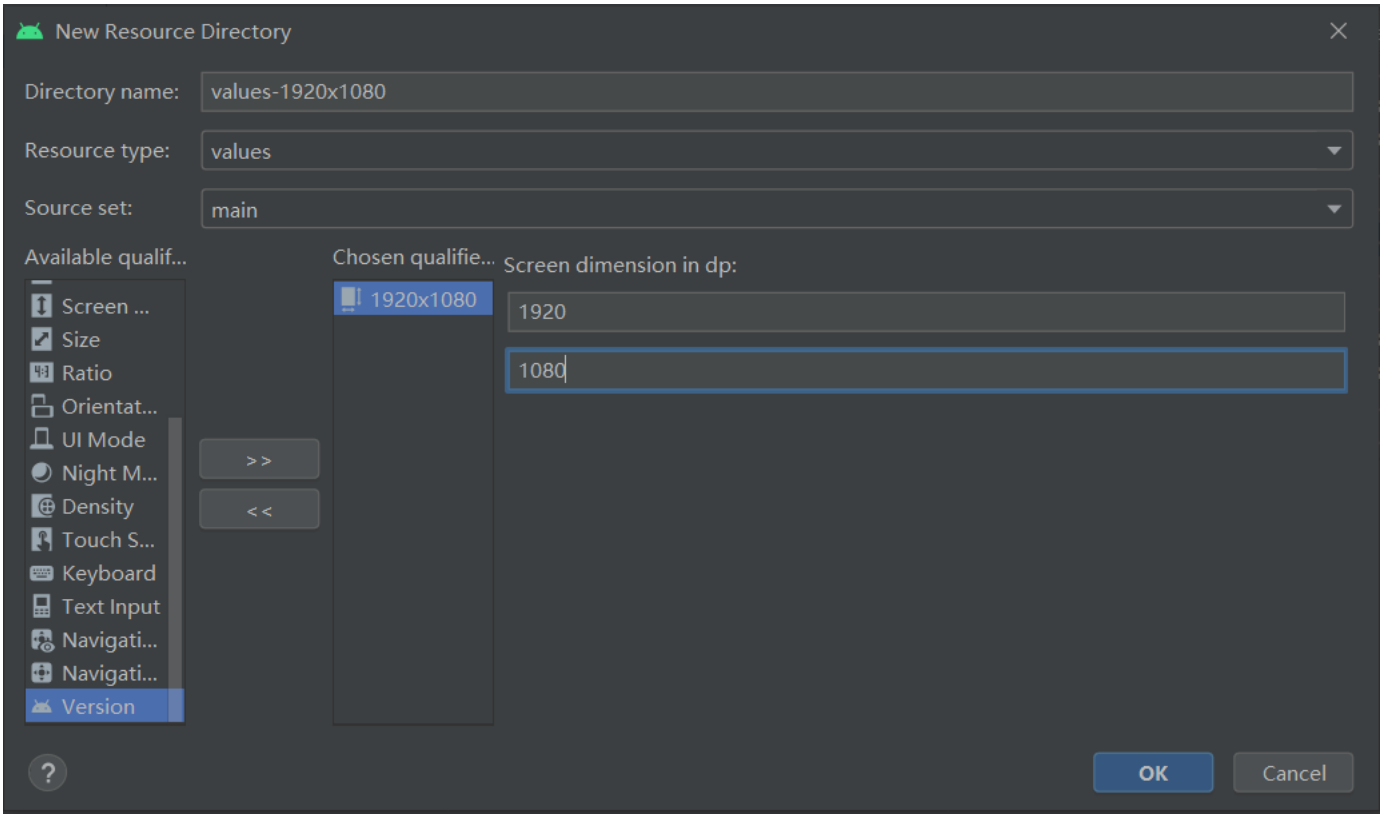
	分辨率	density	densityDpi	scaledDensity	xdpi	ydpi	物理尺寸
Pixel 2	1080 x 1920	2.625	420	2.625	420	420	5.0
Mi Note 3	1080 x 1920	2.75	440	2.75	403	403	5.5
Mi Note 3（大字体）	1080 x 1920	2.75	440	3.85	403	403	5.5

> 首先，
> 1080/2.625=411dp
> 1080/2.75=393dp



dimension适配

在AndroidStudio中创建资源文件夹时，IDE会提醒添加限定符，其中有一个就是限定符就是Dimension：



如上图所示，假设我们填上了1920和1080，生成的文件夹就是values-1920x1080。

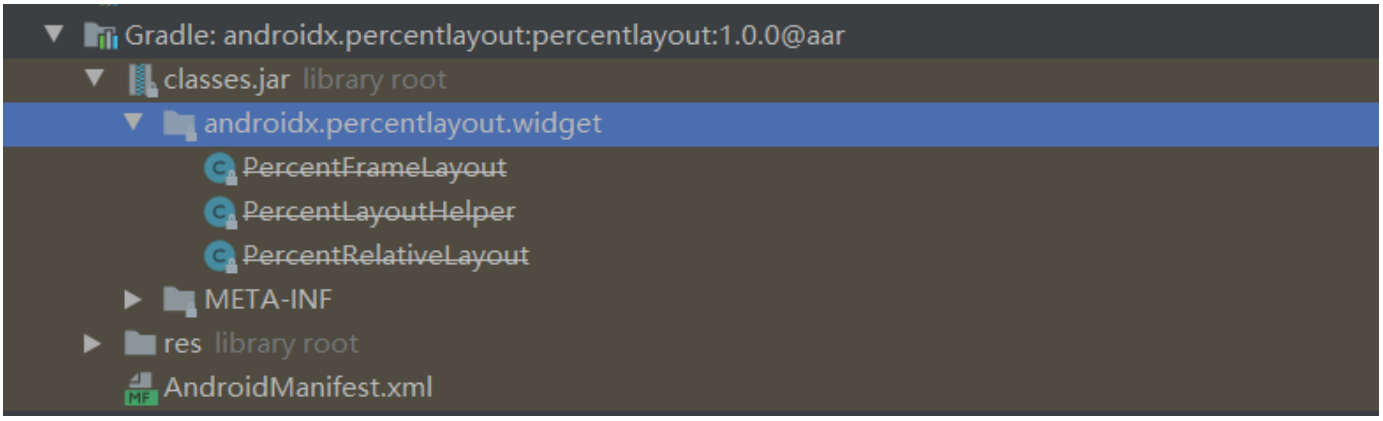
关于该方案的使用见Android适配--最详细的限定符屏幕适配方案解析 附带values-Dimens文件生成工具
关于该限定符的匹配策略和规则见被误用的屏幕分辨率限定符

百分比布局方案

通过子控件设置的百分比属性，动态计算子控件占据父控件的空间百分比，从而达到一个适配的目的。

官方只提供了两个百分比布局，分别为PercentFrameLayout和PercentRelativeLayout，不过现在都已经废弃，主要还是因为需要在onMeasure()中进行子控件的动态计算，耗性能，官方又提供了性能更加的ConstraintLayout，故而percentlayout库基本还未使用就已经结束了生命周期。

percentlayout库的代码结构也很简单，只有3个类：



基本使用

```
implementation "androidx.percentlayout:percentlayout:1.0.0"
```

```
    <androidx.percentlayout.widget.PercentRelativeLayout
        android:layout_width="match_parent"
        android:layout_marginTop="20dp"
        android:layout_height="50dp">

        <TextView
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:background="@android:color/holo_green_dark"
            android:gravity="center"
            android:text="100%"
            app:layout_heightPercent="100%"
            app:layout_widthPercent="100%"
            android:textColor="@android:color/white"
            android:textSize="16sp" />

    </androidx.percentlayout.widget.PercentRelativeLayout>
```

适配效果

源码分析（PercentRelativeLayout）

主要是一个动态计算的过程

PercentRelativeLayout#onMeasure

```
1    private final PercentLayoutHelper mHelper = new PercentLayoutH
  elper(this);
2
3    @Override
4     protected void onMeasure(int widthMeasureSpec, int heightMeas
  ureSpec) {
5             // 调整子控件的尺寸大小
6         mHelper.adjustChildren(widthMeasureSpec, heightMeasureSpe
  c);
7         super.onMeasure(widthMeasureSpec, heightMeasureSpec);
8         if (mHelper.handleMeasuredStateTooSmall()) {
9             super.onMeasure(widthMeasureSpec, heightMeasureSpec);
```

```
        }
    }
```

PercentLayoutHelper#adjustChildren

```
   /**
     * Iterates over children and changes their width and height to one calculated from percentage
     * values.
     * 迭代所有的子控件，动态计算他们宽高对应的百分比值
     * @param widthMeasureSpec Width MeasureSpec of the parent ViewGroup.
     * @param heightMeasureSpec Height MeasureSpec of the parent ViewGroup.
     */
    public void adjustChildren(int widthMeasureSpec, int heightMeasureSpec) {
        // 计算子控件的可用空间，即父View的宽度-左右padding
        int widthHint = View.MeasureSpec.getSize(widthMeasureSpec) - mHost.getPaddingLeft()
                - mHost.getPaddingRight();
        int heightHint = View.MeasureSpec.getSize(heightMeasureSpec) - mHost.getPaddingTop()
                - mHost.getPaddingBottom();
        for (int i = 0, N = mHost.getChildCount(); i < N; i++) {
            // mHost就是PercentRelativeLayout本身
            View view = mHost.getChildAt(i);
            ViewGroup.LayoutParams params = view.getLayoutParams();
            if (params instanceof PercentLayoutParams) {
                PercentLayoutInfo info =
                        ((PercentLayoutParams) params).getPercentLayoutInfo();
                // 区分要不要计算margin
                if (info != null) {
                    if (params instanceof ViewGroup.MarginLayoutParams) {
                        info.fillMarginLayoutParams(view, (ViewGr
```

```
oup.MarginLayoutParams) params,
                                widthHint, heightHint);
                    } else {
                        info.fillLayoutParams(params, widthHint,
heightHint);
                    }
                }
            }
        }
```

PercentLayoutHelper#fillMarginLayoutParams

```
        /**
         * Fills the margin fields of the passed {@link ViewGrou
p.MarginLayoutParams} object based
         * on currently set percentage values and the current lay
out direction of the passed
         * {@link View}.
         */
        public void fillMarginLayoutParams(View view, ViewGroup.M
arginLayoutParams params,
                int widthHint, int heightHint) {
            // 设置View宽高
            fillLayoutParams(params, widthHint, heightHint);

            // 分别根据计算margin的百分比来计算具体的值

            if (leftMarginPercent >= 0) {
                params.leftMargin = Math.round(widthHint * leftMa
rginPercent);
            }
            if (topMarginPercent >= 0) {
                params.topMargin = Math.round(heightHint * topMar
ginPercent);
            }
            if (rightMarginPercent >= 0) {
                params.rightMargin = Math.round(widthHint * right
```

```
MarginPercent);
            }
            if (bottomMarginPercent >= 0) {
                params.bottomMargin = Math.round(heightHint * bottomMarginPercent);
            }

        }
```

PercentLayoutHelper#fillLayoutParams

```
        /**
         * Fills the {@link ViewGroup.LayoutParams#width} and {@link ViewGroup.LayoutParams#height}
         * fields of the passed {@link ViewGroup.LayoutParams} object based on currently set
         * percentage values.
         通过设置的百分比值填充父View的区域
         */
        public void fillLayoutParams(ViewGroup.LayoutParams params, int widthHint, int heightHint) {
            ......

            // 计算百分比对应的宽度
            if (widthPercent >= 0) {
                params.width = Math.round(widthHint * widthPercent);
            }
            // 计算百分比对应的高度
            if (heightPercent >= 0) {
                params.height = Math.round(heightHint * heightPercent);
            }
            // aspectRatio是根据其中一个维度的值来计算另一维度的值，也就是宽高比
            // 比如设置宽度为300dp，aspectRatio=178%，也就是16:9，高度会调整为宽度的16/9
            if (aspectRatio >= 0) {
```

```
                if (widthNotSet) {
                    params.width = Math.round(params.height * aspectRatio);
                    // Keep track that we've filled the width based on the height and aspect ratio.
                    mPreservedParams.mIsWidthComputedFromAspectRatio = true;
                }
                if (heightNotSet) {
                    params.height = Math.round(params.width / aspectRatio);
                    // Keep track that we've filled the height based on the width and aspect ratio.
                    mPreservedParams.mIsHeightComputedFromAspectRatio = true;
                }
            }
        }
```

今日头条方案

布局中的dp转换

```
    public static float applyDimension(int unit, float value,
                                       DisplayMetrics metrics)
    {
        switch (unit) {
        case COMPLEX_UNIT_PX:
            return value;
        case COMPLEX_UNIT_DIP:
            return value * metrics.density;
        case COMPLEX_UNIT_SP:
            return value * metrics.scaledDensity;
        case COMPLEX_UNIT_PT:
            return value * metrics.xdpi * (1.0f/72);
        case COMPLEX_UNIT_IN:
            return value * metrics.xdpi;
        case COMPLEX_UNIT_MM:
```

```
            return value * metrics.xdpi * (1.0f/25.4f);
        }
        return 0;
    }
```

图片缩放使用的目标像素

```
    @Nullable
    public static Bitmap decodeResourceStream(@Nullable Resources res, @Nullable TypedValue value,
            @Nullable InputStream is, @Nullable Rect pad, @Nullable Options opts) {
        validate(opts);
        if (opts == null) {
            opts = new Options();
        }

        if (opts.inDensity == 0 && value != null) {
            final int density = value.density;
            if (density == TypedValue.DENSITY_DEFAULT) {
                opts.inDensity = DisplayMetrics.DENSITY_DEFAULT;
            } else if (density != TypedValue.DENSITY_NONE) {
                opts.inDensity = density;
            }
        }
        // inTargetDensity: 此位图将绘制到的目标的像素密度
        if (opts.inTargetDensity == 0 && res != null) {
            opts.inTargetDensity = res.getDisplayMetrics().densityDpi;
        }

        return decodeStream(is, pad, opts);
    }
```

由上两段代码可知：

1. px = dp * density（density是缩放因子）
2. 图片缩放因素为densityDpi

头条的方案就是动态修改DisplayMetrics的density和densityDpi，已达到适配的目的。例如：就上表中，Pixel 2的宽度是411dp，Mi Note 3的宽度为393dp，那么，如果我们以Mi Note 3为设计图，那么就是说对于Mi Note 3来说是适配的。只要能做到将Pixel 2对应的density也变成2.75的话，那么Pixel 2也就失陪了。

怎样修改Pixel 2的目标density和densityDpi呢？在setContentView之前调用：

```
1  public class DensityUtils {
2      // 以Mi Note 3为基准
3      // 698 * 393
4      private static final int BASE_WIDTH = 393;
5
6      public static void updateDensity(@NonNull Activity activity,
   @NonNull Application application) {
7
8          DisplayMetrics displayMetrics = application.getResources(
   ).getDisplayMetrics();
9          // targetDensity：表示将当前设备的宽修改为基准设计图的宽之后，dens
   ity的最新值
10         float targetDensity = displayMetrics.widthPixels * 1.0f /
   BASE_WIDTH;
11         // targetDensityDpi：表示将当前设备的宽修改为基准设计图的宽之后，d
   ensity的最新dpi
12         int targetDensityDpi = (int) (DisplayMetrics.DENSITY_DEFA
   ULT * targetDensity);
13
14         // 分别赋值给回去Application和Activity的DisplayMetrics对象，这
   样在TypedValue.applyDimension()
15         // 中进行单位换算时，就可以得到合适的换算值
16         displayMetrics.density = displayMetrics.scaledDensity = t
   argetDensity;
17         displayMetrics.densityDpi = targetDensityDpi;
18
19         DisplayMetrics metrics = activity.getResources().getDispl
   ayMetrics();
20         metrics.density = displayMetrics.scaledDensity = targetDe
```

```
    nsity;
21          metrics.densityDpi = targetDensityDpi;
22      }
23  }
```

适配效果

适配前

适配后

参考一种极低成本的Android屏幕适配方式

参考资料

屏幕兼容性概览

应用资源概览

Android 如何查找最佳匹配资源