**适配分析**

**基本概念**

* 屏幕尺寸：按屏幕对角测量的实际物理尺寸。单位是英寸，1英 寸=2.54厘米。
* 显示分辨率：屏幕上物理像素的总数。单位px。
* 图像分辨率：单位英尺中所包含的的像素数。做设计的同学经常用到
* 网点密度（**DPI**）：

在纸质媒介时代，我们常用网点密度（Dot Per Inch，简称DPI）来描述印刷品的打印精度。DPI常用来描述设备参数（如扫描仪和打印机），例如：我设置了打印分辨率为96DPI，那么打印机在打印过程中每英寸（inch）的长度，打印了96个点（dot）。

* 像素密度（**PPI**）：

像素密度（Pixel Per Inch，简称PPI）常用于“屏幕显 示”的描述，用来描述每英寸的像素点数数量。

假设屏幕尺寸（对角线）为z英寸，宽为a英寸，高为b英寸， 分辨率为X像素和Y像素，像素密度为p( 像素 / 英寸 )那么:

➀计算公式：p = X / a =Y / b ( 像素 / 英寸 )；

假如我们只知道z、X、Y的值，p的值怎么计算呢？

根据勾股定理我们知道z²=a²+b²，那么将➀的公式带入到 这个勾股定理中得到：

➁计算公式：p =/ z

注意：根据勾股定理，这个公式中X²+Y²开根号后等于对角线 上的像素点数，然后除以对角线的物理长度就是屏幕 的像素密度。这个理解思路是错误的，因为对角线上的 像素点数根本就不能代表屏幕的像素密度PPI。举个例 子3\*3的像素点阵对角线有3个像素点，4\*4的像素点 阵有4个像素点，以此类推n\*n的对角线有n个像素 点，对角线 的像素点数与他们的横向和纵向的像素 点数相等，假如屏幕的尺寸为3英寸，那么由➁公式得 到它正确的PPI = (\*n) / 3 而不是n / 3。

* 密度无关像素Density Independent Pixels:

用于以密度无关方式表示布局维度或位置，缩写为dip或者 dp。

密度无关像素等于160dpi屏幕上的一个物理像素，这是系统 为“中”密度屏幕假设的基线密度。它的含义是：将一英寸分 为160份，可以理解为是160dp，在各个不同的像素密度的屏 幕上每1dp包含的像素点数不同，dp单位与px的转换如下：



* dp单位转换为px单位：

在某些情况下，您需要以 dp 表示尺寸，然后将它们转换为 像素。设想一个在用户 手指移动至少 16 像素之后可以识别滚动或滑动手势的应用。在基线屏幕上，用户必须移动 16 pixels / 160 dpi（等于一英寸的 1/10 或 2.5 毫米），然后才会识别该手势。在 具有高密度显示屏 (240dpi) 的设备上，用户必须移动 16 pixels / 240 dpi（等于 一英寸的 1/15 或 1.7 毫米）。此距离更短，应用因此 似乎对用户更灵敏。

要修复此问题，手势阈值必须在代码中以 dp 表示，然后 转换为实际像素。例如：

// The gesture threshold expressed in dp  
private static final float GESTURE\_THRESHOLD\_DP = 16.0f;  
  
// Get the screen's density scale  
final float scale = [getResources()](https://developer.android.com/reference/android/content/ContextWrapper.html" \l "getResources()).[getDisplayMetrics()](https://developer.android.com/reference/android/content/res/Resources.html" \l "getDisplayMetrics()).density;  
// Convert the dps to pixels, based on density scale  
mGestureThreshold = (int) (GESTURE\_THRESHOLD\_DP \* scale + 0.5f);  
  
// Use mGestureThreshold as a distance in pixels...

[DisplayMetrics.density](https://developer.android.com/reference/android/util/DisplayMetrics.html" \l "density) 字段根据当前屏幕密度指定 将 dp 单位转换为像素必须使用的缩放系数。 在中密度屏幕上，[DisplayMetrics.density](https://developer.android.com/reference/android/util/DisplayMetrics.html" \l "density) 等于 1.0；在高密度屏幕上，它等于 1.5；在超高密度屏幕上，等于 2.0； 在低密度屏幕上，等于 0.75。此数字是一个系数，应用其乘以 dp 单位以获取用于当前屏幕的实际像素数。（然后在转换时加上 0.5f，将该数字四舍五入到最接近的整数。）如需了解 详细信息，请参阅 [DisplayMetrics](https://developer.android.com/reference/android/util/DisplayMetrics.html) 类。

但是，在一些手势事件中不能这么判断，而应使用 [ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration.html) 中的预缩放配置值。

#### 使用预缩放的配置值

您可以使用 [ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration.html) 类访问 Android 系统使用的通常距离、 速度和时间。例如， 使用 [getScaledTouchSlop()](https://developer.android.com/reference/android/view/ViewConfiguration.html" \l "getScaledTouchSlop()) 可获取框架用作滚动阈值的距离（像素）：

private static final int GESTURE\_THRESHOLD\_DP = ViewConfiguration.get(myContext).getScaledTouchSlop();

[ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration.html) 中以 getScaled 前缀 开头的方法确定会返回不管当前屏幕密度为何都会正常显示的 像素值。

**支持的屏幕范围**

Android将实际的屏幕尺寸和密度分类如下：

四种通用的尺寸：小、正常、大、超大

六种通用密度：ldpi（低）： ~ 120dpi

mdpi（中）：~ 160dpi

hdpi（高）： ~ 240dpi

xhdpi（超高）：~ 320dpi

xxhdpi（超超高）：~ 480dpi

xxxhdpi（超超超高）：~ 640dpi

在为不同的尺寸设计UI时，您会发现每种设计都会需要最 小空间。因此上诉通用的屏幕尺寸都关联了系统定义的最低屏 幕分辨率。这些尺寸以“dp”单位表示，即在定义布局时应该 使用相同的单位。这样系统无需担心屏幕密度的变化：

超大屏幕至少为 960dp x 720dp

大屏幕至少为 640dp x 480dp

正常屏幕至少为 470dp x 320dp

小屏幕至少为 426dp x 320dp

**如何支持多种屏幕**

* Android能针对不同的屏幕以适当的方式渲染应用的布局和位图。但是我们为了更适当的处理不同的屏幕配置，还应该：

1. 在[<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html)中显示的声明您的应用支持哪些屏幕尺寸。

<supports-screens android:[resizeable](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "resizeable)=["true"| "false"]  
                  android:[smallScreens](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "small)=["true" | "false"]  
                  android:[normalScreens](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "normal)=["true" | "false"]  
                  android:[largeScreens](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "large)=["true" | "false"]  
                  android:[xlargeScreens](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "xlarge)=["true" | "false"]  
                  android:[anyDensity](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "any)=["true" | "false"]  
                  android:[requiresSmallestWidthDp](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "requiresSmallest)="*integer*"  
                  android:[compatibleWidthLimitDp](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "compatibleWidth)="*integer*"  
                  android:[largestWidthLimitDp](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "largestWidth)="*integer*"/>

详细见：<https://developer.android.com/guide/topics/manifest/supports-screens-element.html>

2、为不同的屏幕尺寸提供不同的布局。

针对不同的屏幕使用配置限定符small，normal，large，xlarge，来提供特定的资源，例如，超大屏幕的布局使用layout-xlarge/。

但是，从Android3.2（api级别13）开始上面的尺寸组已经弃用，目前使用sw<N>dp配置限定符来定义布局资源可用的最小宽数。例如，多窗格平板电脑布局至少需要600dp的屏幕宽度，应将其放在layout-sw600dp/中

1. 为不同的屏幕密度提供不同的位图和可绘制对象。

默认Android会缩放位图可绘制对象（.png、.jpg、.gif）

和九宫格可绘制对象（ .9.png ），它们以适当的物理尺寸

显示在设备上。

针对不同屏幕密度使用的配置限定符包括 ldpi（低）、mdpi（中）、 hdpi（高）、xhdpi（超高）、xxhdpi （超超高）和 xxxhdpi（超超超高），例如，高密度的位图应该使用drawable-hdpi/

* 使用配置限定符：

Android支持多种配置限定符，可让系统根据当前屏幕屏幕的特性选择备用资源。配置限定符是可以附加到android项目中资源目录的字符串。

使用：1、在项目的res/目录下新建一个目录，并使用以下格式命名：<resources\_name>-<qualifier>

<resources\_name>:是标准资源名称（例如，layout，drawable）

<qualifier>:用于指定要使用这些资源的屏幕配置。如下：



更多详细可参看https://developer.android.com/guide/topics/resources/providing-resources.html#ResourceTypes

* 设计代替布局和可绘制对象

替代布局：为了更好的利用小屏幕或者大屏幕的空间，我们需要 决定是否需要提供不同屏幕尺寸的替代布局。

替代可绘制对象：

要为不同的屏幕密度创建替代位图可绘制资源，应遵循六种通用密度之间的缩放比例：3:4:6:8:12:16的缩放比率。

**声明适用于 Android 3.2 的平板电脑布局**

详情见：<https://developer.android.com/guide/practices/screens_support.html#DeclaringTabletLayouts>

**最佳做法**

1. 在XML文件指定尺寸时应使用wrap\_content、match\_parent 或 dp 单位。
2. 不要再代码中使用硬编码像素值。

由于性能的原因和简化代码的需要，Android系统使用像素作为尺寸或坐标 值的标准单位。这意味着视图的尺寸始终以像素表示，但始终基于当前的屏 幕密度。例如：myView.getWidth()返回的是10，但是在更高的屏幕密度上返回的可能是15。

1. 不要使用AbsoluteLayout。
2. 使用尺寸和特定密度资源。

**其他密度注意事项**

1. 资源的预缩放

当系统加载应用中图片时，假如根据当前屏幕密度没有在相应的资源文件夹中加载到正确的资源，系统将会加载默认资源，并向上或者向下扩展，并且会执行预缩放以将位图调整至适应当前屏幕密度的大小。

如果加载的是预缩放的资源，那么系统会返回预缩放后的尺寸值。

例如：针对mdpi屏幕以50\*50像素设计的位图在hdpi的屏幕上将扩展至75\*75像素，并且系统会返回75\*75的尺寸。

1. 像素和坐标的自动缩放

应用可通过在清单中将 [android:anyDensity](https://developer.android.com/guide/topics/manifest/supports-screens-element.html" \l "any) 设置为 "false" 或者通过将 [inScaled](https://developer.android.com/reference/android/graphics/BitmapFactory.Options.html" \l "inScaled) 设置为 "false" 对 [Bitmap](https://developer.android.com/reference/android/graphics/Bitmap.html) 编程来停用预缩放。在此情况下，系统在绘制时会自动缩放任何绝对的像素坐标和像素 尺寸值。缩放的目的是确保像素定义的屏幕元素仍 以它们在基线屏幕密度 (mdpi) 下的大致相同物理尺寸显示。系统会对应用透明地处理此缩放，并且 向应用报告缩放后的像素尺寸，而不是物理像素尺寸。通常，****不应停用预缩放****。

注：在 Android 3.0 的更高版本中，由于图形框架的改进，应该 觉察不出预缩放的位图 与自动缩放的位图之间 的差异

**设计图浅析**

一般情况下设计师是根据市面上主流的显示分辨率来定义设计图的显示分辨率，现在主流的720\*1280。他们不用考虑手机的像素密度ppi。所以我们只需要关注设计图的显示分辨率和设计图上的标注就可以。

有一个小拓展就是，图像分辨率（看顶部定义），一般设计师在PS中会设置图像分辨率为72px/inch( 像素每英寸 )，这个值也是代表每英寸的像素点数，那么跟我们平时说的ppi有什么关系呢？没有关系！！但是我们可以根据这个值计算出设计图的尺寸：

根据，显示分辨率 = 720 px \* 1280 px，图像分辨率 = 72 px/inch

那么我们将设计图在电脑上以100%的比例展示，他的物理宽度正好是width = 720 / 72 = 10 inch = 25.4cm。

但是这依旧跟我们没有任何的毛线关系。

切图：

一般的手机分辨率和所运行屏幕的ppi是匹配的

320\*1=320 对应屏幕mdpi（160dpi） 屏幕分辨率320\*480px

320\*1.5=480 对应屏幕hdpi（240dpi） 屏幕分辨率480\*800px

320\*2.25=720对应屏幕xhdpi（320dpi） 屏幕分辨率720\*1280px

320\*3.375=1080对应屏幕xxhdpi（480dpi） 屏幕分辨1080\*1920px

其中屏幕密度的比例为1：1.5：2：3。

设计图的选择影响切图的尺寸，例如，选择720\*1280px分辨率的设计图，那么在这个设计图上切下来的原图就是xhdpi的ppi的切图，也就是@2x的图，假如想要获取@3x的图则是在@2x的切图上扩大1.5倍即可。

Android drawable文件夹选择图片逻辑是根据文件夹代表的密度由高到低选择的。