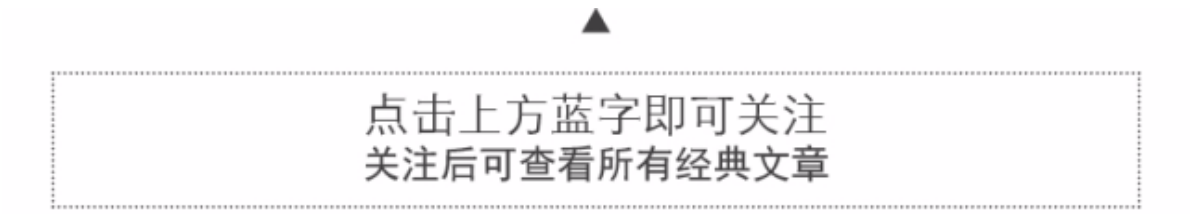
Android 屏幕适配方案

ghroosk



近日作家六六发布微博称，其通过百度查一个上海美国领事馆官网的地址，然后发现搜索结果中有比较多的骗子广告，并@李彦宏质疑他是做搜索引擎还是做骗子首领。对于作家六六的吐槽，百度回应称向六六及广大网友表示诚挚歉意。另外，百度表示，为了更好地提升用户体验，将下线与“上海美国领事”相关的代办签证等广告，并对上海美国领事馆官网及相关品牌词进行品牌保护。

明天就是周六啦，提前祝大家周末愉快！

本篇来自 ghroosk 的投稿，总结了几篇精彩的Android屏幕适配文章，希望对大家有所帮助！

ghroosk 的博客地址：

https://juejin.im/user/5861e1d461ff4b00630bf77c

本文为自身的总结与结合其他文章引用而成，分别为：

- wangwangli6：

Android开发：最全面、最易懂的Android屏幕适配解决方案

https://blog.csdn.net/wangwangli6/article/details/63258270

- jiashuai94：

安卓屏幕完美适配方案——独家秘笈

https://blog.csdn.net/jiashuai94/article/details/77639511

- 司小三石：

android 屏幕适配的总结，适合面试

https://blog.csdn.net/lanxingfeifei/article/details/52161833

- 宇宝守护神：

ImageView的scaleType的属性理解

https://blog.csdn.net/qq_34902522/article/details/76682293

- 秦子帅：

Android刘海屏适配方案

https://www.jianshu.com/p/62c6625db7ab

（暂未总结刘海屏适配，如有需要请查看这篇文章）

- 自身的思考&实践

由于Android系统的开放性，任何用户、开发者、硬件厂商、运营商都可以对Android系统和硬件进行定制，修改成他们想要的样子。 那么这种“碎片化”到达什么程度呢？

以上每一个矩形都代表一种机型，且它们屏幕尺寸、屏幕分辨率大相径庭。随着Android设备的增多，设备碎片化、系统碎片化、屏幕尺寸碎片化、屏幕碎片化的程度也在不断加深。

备注：

1. Android系统碎片化：基于Google原生系统，小米定制的MIUI、魅族定制的flyme、华为定制的EMUI等等；
2. Android机型屏幕尺寸碎片化：5寸、5.5寸、6寸等等；
3. Android屏幕分辨率碎片化：320x480、480x800、720x1280、1080x1920等；

当Android系统、屏幕尺寸、屏幕密度出现碎片化的时候，就很容易出现同一元素在不同手机上显示不同的问题。试想一下这么一个场景： 为4.3寸屏幕准备的UI设计图，运行在5.0寸的屏幕上，很可能在右侧和下侧存在大量的空白；而5.0寸的UI设计图运行到4.3寸的设备上，很可能显示不下。

为了保证用户获得一致的用户体验效果,使得某一元素在Android不同尺寸、不同分辨率的、不同系统的手机上具备相同的显示效果，能够保持界面上的效果一致,我们需要对各种手机屏幕进行适配！

1、像素（px）：

- 含义：通常所说的像素，就是CCD/CMOS上光电感应元件的数量，一个感光元件经过感光，光电信号转换，A/D转换等步骤以后，在输出的照片上就形成一个点，我们如果把影像放大数倍，会发现这些连续色调其实是由许多色彩相近的小方点所组成，这些小方点就是构成影像的最小单位“像素”（Pixel）。简而言之，**像素就是手机屏幕的最小构成单元。**
- 单位：px（pixel），1px = 1像素点 一般情况下UI设计师的设计图会以px作为统一的计量单位。

2、分辨率：

- 含义：手机在横向、纵向上的像素点数总和 一般描述成 宽*高 ，即横向像素点个数 * 纵向像素点个数（如1080 x 1920）。
- 单位：px（pixel），1px = 1像素点

3、屏幕尺寸（in）：

- 含义：手机对角线的**物理尺寸**
- 单位 英寸（inch），一英寸大约2.54cm 常见的尺寸有4.7寸、5寸、5.5寸、6寸

4、屏幕像素密度（dpi）：

- 含义：每英寸的像素点数。 例如每英寸内有160个像素点，则其像素密度为160dpi。
- 单位：dpi（dots per inch）
- 计算公式： **像素密度 = 像素 / 尺寸 （dpi = px / in）**
- 标准屏幕像素密度（mdpi）： 每英寸长度上还有160个像素点（160dpi），即称为标准屏幕像素密度（mdpi）。

密度类型	代表的分辨率（px）	屏幕像素密度（dpi）
低密度（ldpi）	240 x 320	120
中密度（mdpi）	320 x 480	160
高密度（hdpi）	480 x 800	240
超高密度（xhdpi）	720 x 1280	320
超超高密度（xxhdpi）	1080 x 1920	480

屏幕尺寸、分辨率、像素密度三者关系

一部手机的分辨率是宽x高，屏幕大小是以寸为单位，那么三者的关系是：

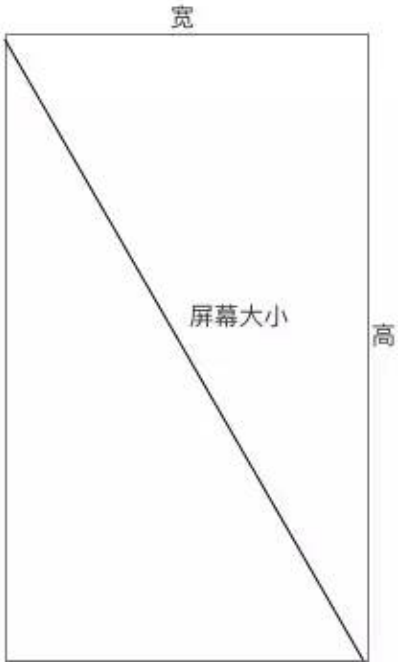
$$密度（单位/dpi）= \frac{\sqrt{（宽^2+高^2）}\ 单位（/px）}{屏幕大小\quad 单位（/inch）}$$

讲解

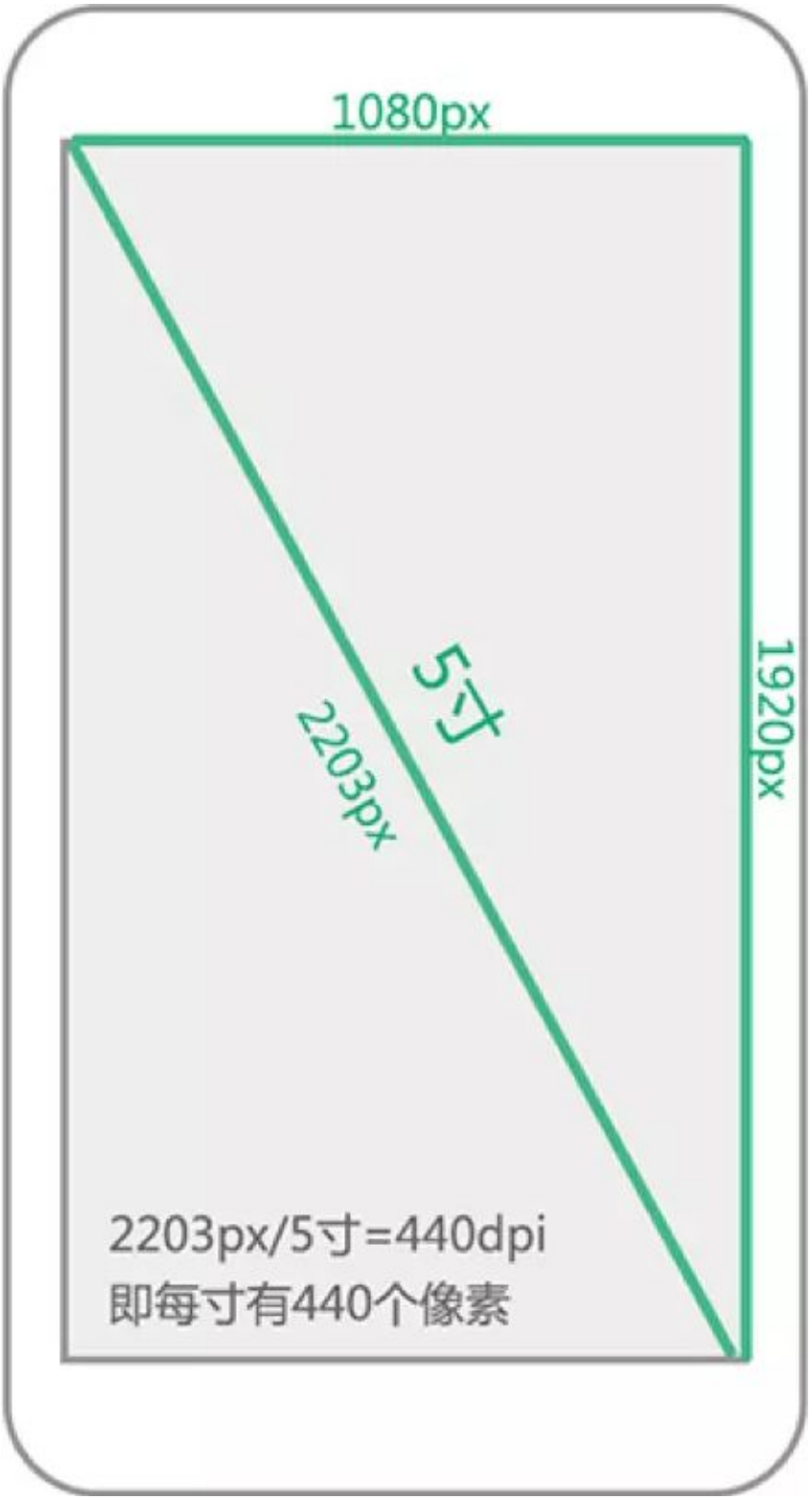
1. 密度即每英寸的像素点

2. 勾股定理求出手机的对角线物理尺寸

3. 再除以屏幕大小即可



假设一部手机的分辨率是1080x1920（px），屏幕大小是5寸



5、密度无关像素（dp）：

- 含义：density-independent pixel，叫dp或dip，与终端上的实际物理像素点无关
- 单位：dp，可以保证在不同屏幕像素密度的设备上显示相同的效果，是安卓特有的长度单位。
- 场景例子：假如同样都是画一条长度是屏幕一半的线，如果使用px作为计量单位，那么在480x800分辨率手机上设置应为240px；在320x480的手机上应设置为160px，二

者设置就不同了；如果使用dp为单位，在这两种分辨率下，160dp都显示为屏幕一半的长度。

- dp与px的转换：1dp = （ dpi / 160 ） * 1px;

密度类型	代表的分辨率（px）	屏幕密度（dpi）	换算
低密度（ldpi）	240 x 320	120	1dp = 0.75px
中密度（mdpi）	320 x 480	160	1dp=1px
高密度（hdpi）	480 x 800	240	1dp=1.5px
超高密度（xhdpi）	720 x 1280	320	1dp=2px
超超高密度（xxhdpi）	1080 x 1920	480	1dp=3px

6、独立比例像素（sp）：

- 含义：scale-independent pixel，叫sp或sip
- 单位：sp，字体大小专用单位 Android开发时用此单位设置文字大小，可根据字体大小首选项进行缩放； 推荐使用12sp、14sp、18sp、22sp作为字体大小，不推荐使用奇数和小数，容易造成精度丢失，12sp以下字体太小。

7、sp 与 dp 的区别：

- dp只跟屏幕的像素密度有关；
- sp和dp很类似但唯一的区别是，Android系统允许用户自定义文字尺寸大小（小、正常、大、超大等等），当文字尺寸是“正常”时1sp=1dp=0.00625英寸，而当文字尺寸是“大”或“超大”时，1sp>1dp=0.00625英寸。类似我们在windows里调整字体尺寸以后的效果——窗口大小不变，只有文字大小改变。

追到android源码，发现系统内部用applyDimension() （路径：android.util.TypedValue.applyDimension()）将所有单位都转换成px 再处理：

```
/**
     * Converts an unpacked complex data value holding a dimension to its final floating
     * point value. The two parameters <var>unit</var> and <var>value</var>
     * are as in {@link #TYPE_DIMENSION}.
     *
     * @param unit The unit to convert from.
```

```
     * @param value The value to apply the unit to.
     * @param metrics Current display metrics to use in the conversion --
     *                supplies display density and scaling information.
     *
     * @return The complex floating point value multiplied by the appropriate
     * metrics depending on its unit.
     */
    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;
}
```

可以发现dp和sp的区别在于density和scaledDensity两个值上；

```
/**
  *  The  logical  density  of  the  display.    This  is  a  scaling  factor  for  the
  *  Density  Independent  Pixel  unit,  where  one  DIP  is  one  pixel  on  an
  *  approximately  160  dpi  screen  (for  example  a  240x320,  1.5"x2"  screen),
  *  providing  the  baseline  of  the  system's  display.  Thus  on  a  160dpi  screen
  *  this  density  value  will  be  1;  on  a  120  dpi  screen  it  would  be  .75;  etc.
  *
  *  <p>This  value  does  not  exactly  follow  the  real  screen  size  (as  given  by
  *  {@link  #xdpi}  and  {@link  #ydpi},  but  rather  is  used  to  scale  the  size  of
  *  the  overall  UI  in  steps  based  on  gross  changes  in  the  display  dpi.    For
  *  example,  a  240x320  screen  will  have  a  density  of  1  even  if  its  width  is
  *  1.8",  1.3",  etc.  However,  if  the  screen  resolution  is  increased  to
  *  320x480  but  the  screen  size  remained  1.5"x2"  then  the  density  would  be
  *  increased  (probably  to  1.5).
  *
```

```
     * @see #DENSITY_DEFAULT
     */
    public float density;
    /**
     * A scaling factor for fonts displayed on the display.  This is the same
     * as {@link #density}, except that it may be adjusted in smaller
     * increments at runtime based on a user preference for the font size.
     */
    public float scaledDensity;
```

屏幕适配问题的本质是使得布局、布局组件在Android不同尺寸、不同分辨率的手机上具备相同的显示效果，下面我将分几个方面来谈谈如何去适配。

3.1 关于布局组件的适配：

- **3.1.1 使用密度无关像素指定尺寸**

由于各种屏幕的像素密度都有所不同，因此相同数量的像素在不同设备上的实际大小也会有所差异，这样使用像素（px）定义布局尺寸就会产生问题。 因此，请务必使用密度无关像素 dp 或独立比例像素 sp 单位指定尺寸。

备注 :在生产过程中，厂家不会完全按照屏幕密度标准去生产Android设备，会在Google的标准周围浮动变化，或是偏离Google的屏幕密度标准比较大，再加上理论计算（开方）造成的误差，实际上使用dp作为单位是不能完完全全的完成适配操作；

- **3.1.2 使用相对布局或线性布局，不要使用绝对布局**

对于线性布局（Linearlayout）、相对布局（RelativeLayout）、帧布局（FrameLayout）、绝对布局（AbsoluteLayout）以及新增的加强版帧布局（CoordinatorLayout）需要根据需求进行选择，没有绝对而言。

但因为RelativeLayout讲究的是相对位置，即使屏幕的大小改变，视图之前的相对位置都不会变化，与屏幕大小无关，灵活性很强，而LinearLayout法准确地控制子视图之间的位置关系，

只能简单的一个挨着一个地排列，所以，对于屏幕适配来说，使用相对布局（RelativeLayout）将会是更好的解决方案，至于绝对布局由于适配性极差，所以极少使用。

- **3.1.3 使用wrap_content、match_parent、权重**

使用 “wrap_content” 和 “match_parent” 尺寸值而不是硬编码的尺寸，系统会自动计算相应的数值，视图就会相应地使用自身所需的空间或填满可用空间，让布局正确适应各种屏幕尺寸和屏幕方向，组件的权重比同理。

- **3.1.4 使用minWidth、minHeight、lines等属性**

很多时候我们显示的数据都是由后台返回的，再由我们加工处理后去适配我们的组件，这些数据的长度我们是无法确定的，而正常情况下我们构思的布局都仅是适用于理想的情况下，为了保证界面的对齐、数据显示完整等等的原因，我们需要在构思布局时增加对组件最小宽高度、行数等属性的设置，确保在特殊的数据下不会破坏我们的整体布局。

- **3.1.5 dimens使用**

组件的长宽我们可以通过dimens来定义，不同的屏幕尺寸可以定义不同的数值，或者是不同的语言显示我们也可以定义不同的数值，因为翻译后的长度一般都不会跟中文的一致。

3.2 关于布局的适配：

以上几种方式可以解决屏幕适配性的问题，但是那些通过伸缩控件来适应各种不同屏幕大小的布局，未必就是提供了最好的用户体验。你的应用程序应该不仅仅实现了可自适应的布局，还应该提供一些方案根据屏幕的配置来加载不同的布局，可以通过配置限定符(configuration qualifiers)来实现。配置限定符允许程序在运行时根据当前设备的配置自动加载合适的资源(比如为不同尺寸屏幕设计不同的布局)。

- **3.2.1 使用Size限定符**

很多应用会在较大的屏幕上实施“双面板”模式，即在一个面板上显示项目列表，而在另一面板上显示对应内容。平板电脑和电视的屏幕已经大到可以同时容纳这两个面板了，但手机屏幕就需要分别显示。因此，我们可以使用以下文件以便实施这些布局：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="match_parent" />
</LinearLayout>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp" />
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

请注意第二种布局名称目录中的 large 限定符。系统会在属于较大屏幕（例如 7 英寸或更大的平板电脑）的设备上选择此布局。系统会在较小的屏幕上选择其他布局（无限定符）。

- **3.2.2 最小宽度限定符**

使用Size限定符有一个问题会让很多程序员感到头疼，large到底是指多大呢？很多应用程序都希望能够更自由地为不同屏幕设备加载不同的布局，不管它们是不是被系统认定为”large”。这就是Android为什么在3.2以后引入了”Smallest-width”限定符。

最小宽度限定符可让您通过指定某个最小宽度（以 dp 为单位）来定位屏幕。例如，标准 7 英寸平板电脑的最小宽度为 600 dp，因此如果您要在此类屏幕上的用户界面中使用双面板（但在较小的屏幕上只显示列表），您可以使用上文中所述的单面板和双面板这两种布局，但您应使用 sw600dp 指明双面板布局仅适用于最小宽度为 600 dp 的屏幕，而不是使用 large 尺寸限定符。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="match_parent" />
</LinearLayout>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

也就是说，对于最小宽度大于等于 600 dp 的设备，系统会选择 layout-sw600dp/main.xml（双面板）布局，否则系统就会选择 layout/main.xml（单面板）布局。

但 Android 版本低于 3.2 的设备不支持此技术，原因是这些设备无法将 sw600dp 识别为尺寸限定符，因此我们仍需使用 large 限定符。这样一来，就会有一个名称为 res/layout-large/main.xml 的文件（与 res/layout-sw600dp/main.xml 一样）。但是没有太大关系，我们将马上学习如何避免此类布局文件出现的重复。

- **3.2.3 使用布局别名**

最小宽度限定符仅适用于 Android 3.2 及更高版本。因此，如果我们仍需使用与较低版本兼容的概括尺寸范围（小、正常、大和特大）。例如，如果要将用户界面设计成在手机上显示单面板，但在 7 英寸平板电脑、电视和其他较大的设备上显示多面板，那么我们就需要提供以下文件：

res/layout/main.xml: 单面板布局 res/layout-large: 多面板布局 res/layout-sw600dp: 多面板布局

后两个文件是相同的，因为其中一个用于和 Android 3.2 设备匹配，而另一个则是为使用较低版本 Android 的平板电脑和电视准备的。

要避免平板电脑和电视的文件出现重复（以及由此带来的维护问题），您可以使用别名文件。例如，您可以定义以下布局：

- res/layout/main.xml，单面板布局
- res/layout/main_twopanes.xml，双面板布局

然后添加这两个文件：

res/values-large/layout.xml:

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

res/values-sw600dp/layout.xml:

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

后两个文件的内容相同，但它们并未实际定义布局。它们只是将 main 设置成了 main_twopanes 的别名。由于这些文件包含 large 和 sw600dp 选择器，因此无论 Android 版本如何，系统都会将这些文件应用到平板电脑和电视上（版本低于 3.2 的平板电脑和电视会匹配 large，版本高于 3.2 的平板电脑和电视则会匹配 sw600dp）。

- **3.2.4 使用屏幕方向限定符**

某些布局会同时支持横向模式和纵向模式，但我们可以通过调整优化其中大部分布局的效果。在新闻阅读器示例应用中，每种屏幕尺寸和屏幕方向下的布局行为方式如下所示：

小屏幕，纵向：单面板，带徽标

小屏幕，横向：单面板，带徽标

7 英寸平板电脑，纵向：单面板，带操作栏

7 英寸平板电脑，横向：双面板，宽，带操作栏

10 英寸平板电脑，纵向：双面板，窄，带操作栏

10 英寸平板电脑，横向：双面板，宽，带操作栏

电视，横向：双面板，宽，带操作栏

因此，这些布局中的每一种都定义在了 res/layout/ 目录下的某个 XML 文件中。为了继续将每个布局分配给各种屏幕配置，该应用会使用布局别名将两者相匹配：

res/layout/onepane.xml:(单面板)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="match_parent" />
</LinearLayout>
```

res/layout/onepane_with_bar.xml:(单面板带操作栏)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:layout_width="match_parent"
                  android:id="@+id/linearLayout1"
                  android:gravity="center"
                  android:layout_height="50dp">
        <ImageView android:id="@+id/imageView1"
                   android:layout_height="wrap_content"
                   android:layout_width="wrap_content"
                   android:src="@drawable/logo"
                   android:paddingRight="30dp"
                   android:layout_gravity="left"
                   android:layout_weight="0" />
        <View android:layout_height="wrap_content"
              android:id="@+id/view1"
              android:layout_width="wrap_content"
              android:layout_weight="1" />
        <Button android:id="@+id/categorybutton"
                android:background="@drawable/button_bg"
                android:layout_height="match_parent"
                android:layout_weight="0"
                android:layout_width="120dp"
                style="@style/CategoryButtonStyle"/>
    </LinearLayout>

    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="match_parent" />
</LinearLayout>
```

res/layout/twopanes.xml:(双面板，宽布局)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

res/layout/twopanes_narrow.xml:(双面板，窄布局)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="200dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>
```

既然我们已定义了所有可能的布局，那就只需使用配置限定符将正确的布局映射到各种配置即可。

现在只需使用布局别名技术即可做到这一点：

res/values/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/onepane_with_bar</item>
    <bool name="has_two_panes">false</bool>
</resources>
```

1446103188926681.jpg

res/values-sw600dp-land/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes</item>
    <bool name="has_two_panes">true</bool>
</resources>
```

1446103198972935.jpg

res/values-sw600dp-port/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/onepane</item>
    <bool name="has_two_panes">false</bool>
</resources>
```

1446103206749499.jpg

res/values-large-land/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes</item>
    <bool name="has_two_panes">true</bool>
</resources>
```

1446103214277258.jpg

res/values-large-port/layouts.xml:

```
<resources>
    <item name="main_layout" type="layout">@layout/twopanes_narrow</item>
    <bool name="has_two_panes">true</bool>
</resources>
```

- **3.2.5 多套layout适配**

res/values/layouts.xml: res/values-sw600dp-land/layouts.xml: res/values-sw600dp-port/layouts.xml: res/values-large-land/layouts.xml: res/values-large-port/layouts.xml:

3.3 关于图片的适配：

- **3.3.1 LOGO 图标**

建议按官方标准准备好各个图标；

屏幕密度	对应的图片大小	图片资源目录
120dip	36px * 36px	mipmap-ldpi
160dip(基准)	48px * 48px	mipmap或者mipmap-mdpi
240dip(1.5倍)	72px * 72px	mipmap-hdpi
320dip (2倍)	96px * 96px	mipmap-xhdpi
480dip (3倍)	144px * 144px	mipmap-xxhdpi
640dip (4倍)	192px * 192px	mipmap-xxxhdpi

- **3.3.2 普通图片和图标**

建议安装官方的密度类型进行切图即可，但一般我们只需xxhdpi或xxxhdpi的切图即可满足我们的需求；

- **3.3.3 自动拉伸位图：Nine-Patch的图片类型**

支持不同屏幕大小通常情况下也意味着，你的图片资源也需要有自适应的能力。例如，一个按钮的背景图片必须能够随着按钮大小的改变而改变。 如果你想使用普通的图片来实现上述功能，你会发现这是难以实现的，因为运行时会均匀地拉伸或压缩你的图片。解决方案是使用nine-patch图片，它是一种被特殊处理过的PNG图片，你可以指定哪些区域可以拉伸而哪些区域不可以。

- **3.3.4 动画、自定义view、shape**

可以使用代码进行控制和展示多种视图，如patch动画替代帧动画。

- **3.3.5 ImageView的ScaleType适配**

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）

3.4 关于代码适配：

在代码中使用Google提供的API对设备的屏幕宽度进行测量，然后按照需求进行设置。

对于当前控件的宽高设置，需要做的操作是首先要获取到该控件的父控件，使用父控件对当前控件的宽高进行设置操作！

```
DisplayMetrics metrics = new DisplayMetrics ();
```

```
getWindowManager().getDefaultDisplay().getMetrics(metrics);

手机对应的宽高：

Constants.screenHeight=  metrics.heightDixels;

Constants.screenWidth=  metrics.widthDixels;

RelativeLayout.LayoutParams=new  RelativeLayout.LayoutParams();

(int)(  Constants.screenHeight*0.5+0.5f);

(int)(  Constants.screenWidth  *0.5+0.5f);
```

在上面的两个计算操作中最后加上0.5f的作用是：进行float强转到int类型的时候会出现都是精度的问题。当使用Java代码进行宽高设置的时候，假如出现320.2dp这样的数据此时直接进行int得到的值是320；但是假如出现320.7这样的数据的时候，由于int的计算规则，会直接强转为320，但是从实际出发，这个时候的值取321更为合适。

所以在计算的最后直接加0.5，这样一来，320.2+0.5=320.7，进行数据的强转操作得到的数据是320，320.7+0.5=321.2，进行数据强转操作得到的数据是321，这样一来得到的数据就和实际预想的更为接近！！

3.5 关于接口配合：

本地加载图片前判断手机分辨率或像素密度，向服务器请求对应级别图片。

经过上面的介绍，相信大家对于屏幕的适配都有了一定的了解，但实际上我们并不会完全去执行上面的全部操作，而是需要根据我们的项目需求去选择最合适的方法去适配。例如说你的产品是针对老年人的，你的字体单位是使用sp还是dp呢？又比如说RelativeLayout和权重能较好的解决适配的问题，但实际上它们消耗更多的性能，如何去衡量性能与适配的度呢？ 知识是死的，人是活的，能灵活运用相关知识方显真本事。

或者　扫一扫　关注我的公众号

阅读原文