|  |
| --- |
|  |
| Android屏幕适配建议文档 |
| Android屏幕适配初探 |

目录

[Android屏幕适配建议文档 3](#_Toc405795903)

[第一章屏幕相关概念 3](#_Toc405795904)

[1.1、屏幕像素px 3](#_Toc405795905)

[1.2、屏幕大小size 3](#_Toc405795906)

[1.3、屏幕密度dpi 3](#_Toc405795907)

[1.4、屏幕方向 3](#_Toc405795908)

[1.5、分辨率 4](#_Toc405795909)

[1.6、独立密度像素DIP（Density-independent Pixels与密度无关的像素）： 4](#_Toc405795910)

[1.7、DPI与DIP的换算 4](#_Toc405795911)

[1.8、点9图片 5](#_Toc405795912)

[第二章为什么会出现Android屏幕适配问题。 7](#_Toc405795913)

[2.1、Android使用版本的多样化 7](#_Toc405795914)

[2.2、Android设备屏幕尺寸和屏幕密度的多样化 8](#_Toc405795915)

[第三章Android设备加载图片资源的流程和原理 10](#_Toc405795916)

[3.1、Android中放置图片资源的文件夹 10](#_Toc405795917)

[3.2、Android中view设置背景图片时查找图片资源的顺序 10](#_Toc405795918)

[3.3、设置view背景图片的过程 11](#_Toc405795919)

[第四章屏幕适配的解决办法 14](#_Toc405795920)

[4.1、适配不同屏幕尺寸的解决办法。 14](#_Toc405795921)

[4.1.1、建议多使用 "wrap\_content" 和 "match\_parent" 14](#_Toc405795922)

[4.1.2、建议多使用RelativeLayout（相对布局） 16](#_Toc405795923)

[4.1.3、建议使用尺寸标示符 18](#_Toc405795924)

[4.1.4、建议使用最小宽度标识符 19](#_Toc405795925)

[4.1.5、建议使用布局别名 20](#_Toc405795926)

[4.1.6、建议使用横竖标识符 21](#_Toc405795927)

[4.1.7、建议使用.9.png图片资源 25](#_Toc405795928)

[4.2、适配不同屏幕密度的解决办法 26](#_Toc405795929)

[4.2.1、建议使用DIP 26](#_Toc405795930)

[4.2.2、建议提供可供选择的位图 27](#_Toc405795931)

[4.3、适配特殊屏幕设备的解决办法 28](#_Toc405795932)

[4.3.1、建议声明该APP只能运行在手持设别上 28](#_Toc405795933)

[4.3.2、建议声明你的应用程序仅适用于平板设别 29](#_Toc405795934)

[4.3.3、建议发布多个版本的APK适配不同设别 31](#_Toc405795935)

[第五章Android图标设计建议 32](#_Toc405795936)

[5.1、Android Icon 多大时合适 32](#_Toc405795937)

[5.2、Android Icon 尺寸 32](#_Toc405795938)

[5.2.1、Android launcher图标 32](#_Toc405795939)

[5.2.2、操作栏，对话框和图标选项卡 33](#_Toc405795940)

[5.2.3、小的上下文 Icons 33](#_Toc405795941)

[5.2.4、通知栏 icons 33](#_Toc405795942)

[5.3、Google Paly的Icon又该如何设计 34](#_Toc405795943)

[5.4、了解Android的图标大小 34](#_Toc405795944)

[5.5、Android用户界面图标大小 35](#_Toc405795945)

[5.6、如何做出精美的Icon 35](#_Toc405795946)

[5.6.1、Android操作栏图标大小 36](#_Toc405795947)

[5.6.2、Android小型/上下文图标大小 37](#_Toc405795948)

[5.6.3、Android通知图标大小 37](#_Toc405795949)

[第六章Android屏幕适配建议综合 38](#_Toc405795950)

[6.1、屏幕适配宗旨 38](#_Toc405795951)

[6.2、屏幕适配建议 38](#_Toc405795952)

# Android屏幕适配建议文档

## 第一章屏幕相关概念

### 1.1、屏幕像素px

是屏幕的物理像素点，与密度相关，密度大了，单位面积上的px会比较多。

### 1.2、屏幕大小size

通过查阅设备信息可以知道设备的物理尺寸，也可以通过计算屏幕对角线的长度得到，单位为Inch，1 英寸= 2.539999918 厘米(公分)。

### 1.3、屏幕密度dpi

屏幕在物理尺寸范围内的像素数量。也通常指DPI(dots per inch)。屏幕密度越小，所包含的像素也就越少。

DPI＝（√（横向分辨率^2+纵向分辨率^2））/屏幕尺寸  
在Android的设计规范中，把屏幕密度划分为6类

屏幕密度等级：

ldpi (low) ~120dpi（基本上已经淘汰了）

mdpi (medium) ~160dpi

hdpi (high) ~240dpi

xhdpi (extra-high) ~320dpi

xxhdpi (extra-extra-high) ~480dpi

xxxhdpi (extra-extra-extra-high) ~640dpi

### 1.4、屏幕方向

屏幕方向有横向和纵向，这两种情况下的屏幕长宽比是不同的。

### 1.5、分辨率

跟电脑分辨率的概念类似，表示屏幕横纵方向的像素数，例如480\*800。  
在为APP进行屏幕适配时，不能只考虑到像素，还有屏幕大小，屏幕密度等等。

分辨率等级：

xlarge screens are at least 960dp x 720dp

large screens are at least 640dp x 480dp

normal screens are at least 470dp x 320dp

small screens are at least 426dp x 320dp

### 1.6、独立密度像素DIP（Density-independent Pixels与密度无关的像素）：

一个基于density的抽象单位，这个和设备硬件有关，通常在开发中设置一些view的宽高推荐用这个，一般情况下，在不同分辨率，都不会有缩放的感觉。在运行时, Android根据使用中的屏幕的实际密度, 透明地处理任何所需dip单位的缩放。

### 1.7、DPI与DIP的换算

dots per inch，就是每英寸多少像素，通过下面公式可以得到。  
DPI＝（√（横向分辨率^2+纵向分辨率^2））/屏幕尺寸

在Android的设计规范中，DPI分成了5个档次：MDPI，HDPI，XHDPI，XXHDPI，XXXHDPI，它们的比例是 2:3:4:6:8

[](http://isux.tencent.com/wp-content/uploads/2014/06/20140630193233262.png)

在MDPI的屏幕(即160DPI)，1dp 和1sp 基本等于 1px 。

* dp 和 px的换算公式：px ＝ dp\*(DPI/160)。
* sp 和 px的换算公式：px ＝ sp\*(DPI/160)。

### 1.8、点9图片

与传统的png 格式图片相比， 9.png 格式图片在图片四周有一圈一个像素点组成的边沿，该边沿用于对图片的可扩展区和内容显示区进行定义。

   这种格式的图片在android 环境下具有自适应调节大小的能力。

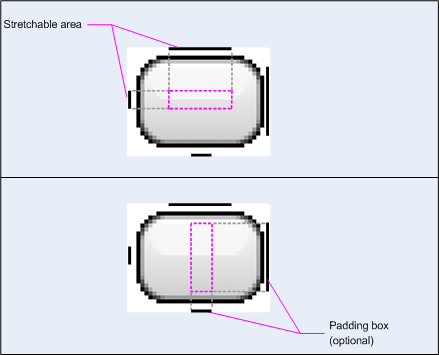
   （1）允许开发人员定义可扩展区域，当需要延伸图片以填充比图片本身更大区域时，可扩展区的内容被延展。

   （2）允许开发人员定义内容显示区，用于显示文字或其他内容

    如下图所示：

    左侧和上方的黑线交叉的部分即可扩展区域

    右侧和下方的黑线交叉的部分即内容显示区



点9图的处理过程和上面的普通png图片是一样的，会根据所放置的资源文件夹和屏幕的像素密度先进行缩放，在显示的时候点9图会再进行局部拉伸，所以如果将带圆角的点9图片放置在低像素密度资源文件夹下，当使用高像素密度设备显示时，图片会先进行放大在进行局部拉伸，这样会导致在放大过程中图片圆角和边缘被拉伸，显示时会变的模糊。

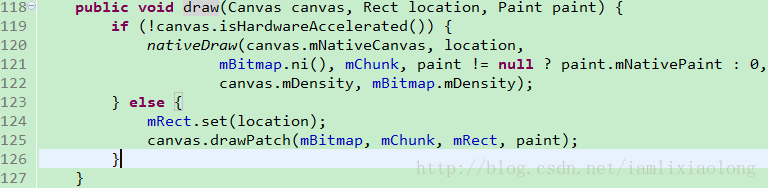
解决方案：

1、尽量将点9图片放置在高像素密度资源文件夹中，这样即使在低像素密度手机上显示时会先对图片进行缩小再进行局部拉伸，但是在低像素密度手机上运行应用时，所有使用点9图片的地方都会对图片进行一次计算缩放，影响性能；

2、针对不同像素密度手机做多套点9图片。

补充：点9图片在缩放过后，如何进行局部拉伸渲染到屏幕上的？

源码跟踪，在View的draw方法中根据Drawable对象将图片作为背景绘制到指定区域中，点9图的实际绘制过程在NinePatch的draw方法中，通过canvas对象调用了本地方法nativeDraw对图片进行了绘制。至于如何绘制局部暂时看不到JNI方法的源码。

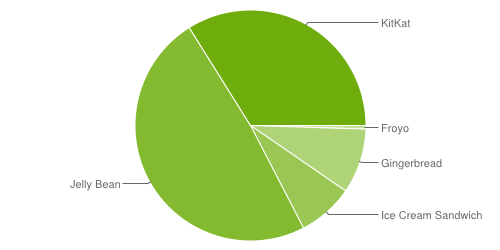


## 第二章为什么会出现Android屏幕适配问题。

### 2.1、Android使用版本的多样化

一下这张表格是反映现有Android版本的使用比重

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Codename | API | Distribution |
| [2.2](https://developer.android.com/about/versions/android-2.2.html) | Froyo | 8 | 0.5% |
| [2.3.3 - 2.3.7](https://developer.android.com/about/versions/android-2.3.3.html) | Gingerbread | 10 | 9.1% |
| [4.0.3 - 4.0.4](https://developer.android.com/about/versions/android-4.0.html) | Ice Cream Sandwich | 15 | 7.8% |
| [4.1.x](https://developer.android.com/about/versions/android-4.1.html) | Jelly Bean | 16 | 21.3% |
| [4.2.x](https://developer.android.com/about/versions/android-4.2.html) | 17 | 20.4% |
| [4.3](https://developer.android.com/about/versions/android-4.3.html) | 18 | 7.0% |
| [4.4](https://developer.android.com/about/versions/android-4.4.html) | KitKat | 19 | 33.9% |



以上数据统计截止到到2014年12月1日。（0.1%以下的样本没有列入以上表格和图表）

**由以上图表分析得现在使用Android4.0以上的用户比重占到了将近90%**

**但是这90%的占有率并不集中在某一版本之上，而是分布于从api15到19的不同版本上这就造成了我们写Android应用是需要去适配不同版本，是其原因之一。**

### 2.2、Android设备屏幕尺寸和屏幕密度的多样化

以下图表提供有关具不同屏幕大小和密度占有比例。Android官方将Android设备的屏幕尺寸和屏幕密度人为的划分为不同的等级。

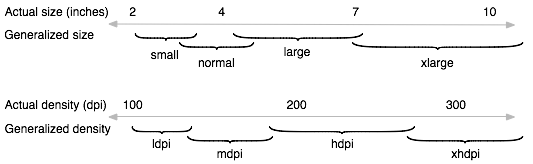
屏幕分辨率等级：

* xlarge screens are at least 960dp x 720dp
* large screens are at least 640dp x 480dp
* normal screens are at least 470dp x 320dp
* small screens are at least 426dp x 320dp

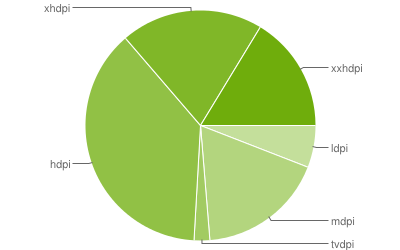
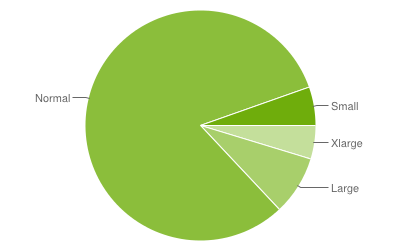
屏幕密度等级：

* ldpi (low) ~120dpi
* mdpi (medium) ~160dpi
* hdpi (high) ~240dpi
* xhdpi (extra-high) ~320dpi
* xxhdpi (extra-extra-high) ~480dpi
* xxxhdpi (extra-extra-extra-high) ~640dpi

大致分布情况统计如下表：



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ldpi | mdpi | tvdpi | hdpi | xhdpi | xxhdpi | **Total** |
| Small | 5.4% |  |  |  |  |  | **5.4%** |
| Normal |  | 9.4% | 0.2% | 36.9% | 18.8% | 16.3% | **81.6%** |
| Large | 0.5% | 4.6% | 2.0% | 0.6% | 0.6% |  | **8.3%** |
| Xlarge |  | 3.8% |  | 0.3% | 0.6% |  | **4.7%** |
| **Total** | **5.9%** | **17.8%** | **2.2%** | **37.8%** | **20.0%** | **16.3%** |  |



以上数据统计截止到2014年12月1日占有率0.1%以下设备没有纳入其中

**由以上数据分析得屏幕分辨率在470dp x 320dp以上的设备占到了将近95%，屏幕密度在240dpi以上的设备占到了将近80%，但是他们同样分布不均，从而导致了我们的Android应用需要去适配不同的屏幕分辨率和不同的屏幕密度的设备**

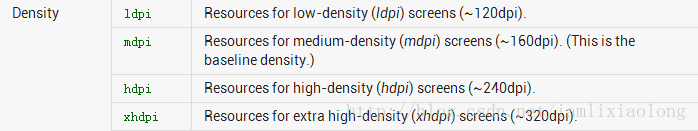
**屏幕尺寸单位是英寸而不是像素，也就说一个英寸在任何分辨率下显示的大小都是一样的，但是像素在密度不同的手机里面显示的实际的大小是不一样的**

## 第三章Android设备加载图片资源的流程和原理

### 3.1、Android中放置图片资源的文件夹

Android中一般有drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi、drawable-xxhdpi等放置图片资源的文件夹，这几个文件夹分别对应的像素密度为：

|  |  |
| --- | --- |
| 文件夹 | 对应的像素密度 |
| drawable-ldpi | 120dpi |
| drawable-mdpi | 160dpi |
| drawable-hdpi | 240dpi |
| drawable-xhdpi | 320dpi |



另外自己可以创建一个默认的drawable文件夹，默认对应的像素密度为160dpi。

### 3.2、Android中view设置背景图片时查找图片资源的顺序

Android中view在setBackground加载图片时，通常会先去设备对应的像素密度的文件夹中去找对应的图片，如果没有找到就会去紧挨着的像素密度较高的文件夹中去找，然后再往上找，如果最高像素密度的文件夹中还没有找到该图片才会向像素密度较低的文件夹中去找。这是加载对应图片的一个查找过程。

比如一个设备的像素密度为240dpi，应用程序有drawable、drawable-ldpi、drawable-mdp、drawable-hdpi、drawable-xhdpi、drawable-xxhdpi六个文件夹，则在一个view设置背景图片时，查找图片的顺序为drawable-hdpi   ===== >  drawable-xhdpi  ====>  drawable-xxhdpi ====>   drawable-mdpi  ====>   drawable  ====>  drawable-ldpi。

这个顺序可以通过一个小的demo自己验证，是android查找图片资源的规则，不是通过几句话就能说明的。

### 3.3、设置view背景图片的过程

源码跟踪：

View view = new View(this);

view.setBackgroundResource(R.drawable.about\_logo);

使用这里是设置一个view的背景图片，setBackgroundResource方法的源码为：

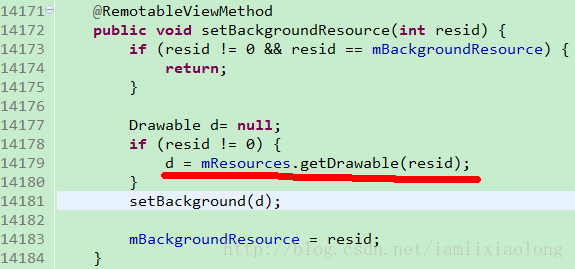


图-2

这个方法里面第14179行代码，红线标注部分，是通过Resources对象根据图片资源resid去获取图片对应的drawable对象，getDrawable方法源码如下：

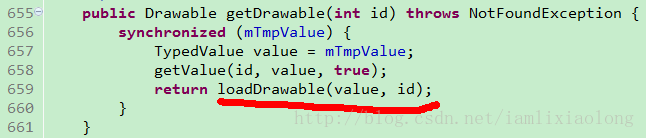


图-3

这个方法里面通过loadDrawable方法返回一个Drawable对象，loadDrawable方法里面传入了一个TypedValue对象，而TypedValue对象是通过getValue方法获得的，这里可以通过代码查看一下TypedValue对象中存放了哪些对应资源图片的信息。

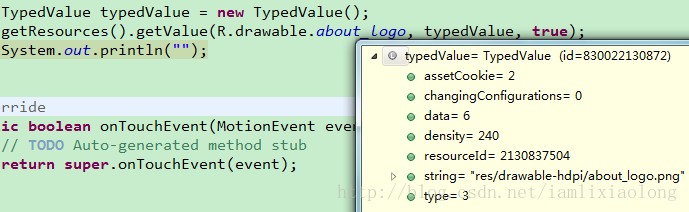


图-4

通过demo代码中，对代码进行debug，发现，根据图片资源resid获取的TypeValued对象中保存的信息主要有density=240和string=“res/drawable-hdpi/about-logo.png”，density是指找到的图片资源所在drawable-hdpi文件夹对应的像素密度，string是图片资源的路径。

实际上loadDrawable方法就是根据这个图片资源的路径去获取到相应的Drawable对象的。此时我是将图片放置到drawable-hdpi文件夹中的，那么如果我将图片移动到drawable-mdpi文件夹中，TypeValued值会一样吗？通过测试，发现如下结果：

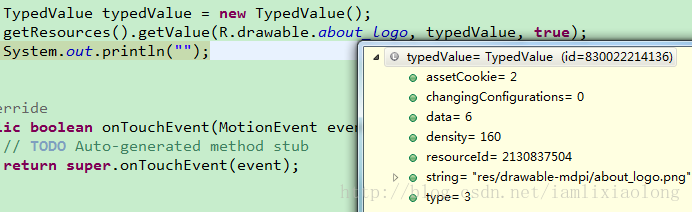


图-5

通过图-4和图-5得出的typedValue信息可知，其获得的相应变量值是不一样的，此时的density=160、string=”res/drawable-mdpi/about-logo.png”，density是drawable-mdpi对应的像素密度，那么同样的图片放置在不同的资源文件夹中，得到的Drawable对象一样吗，通过证明，它们是不一样的。

about\_logo.png原始图片大小为138\*64像素，在同样的480\*800像素240dpi的模拟器上运行，其得到的Drawable对象信息如下：

图片放置在drawable-mdpi文件夹下：

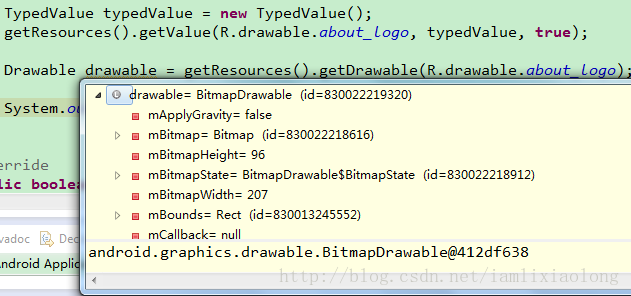


图-6

图片放置在drawable-hdpi文件夹下：

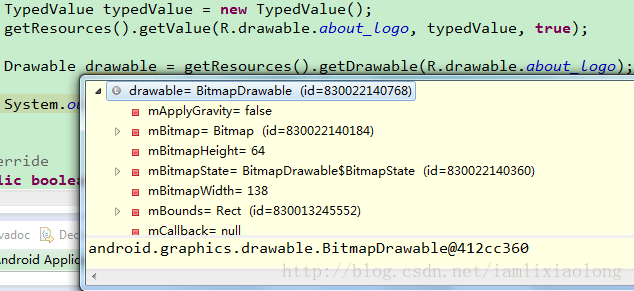


图-7

通过代码测试得出如下数据：

|  |  |  |  |
| --- | --- | --- | --- |
| 图片放置的文件夹 | 对应像素密度 | 设备像素密度 | 得到的图片对应的Bitmap的宽高值 |
| drawable-mdpi | 160dpi | 240dpi | 207\*96 |
| drawable-hdpi | 240dpi | 240dpi | 138\*64 |

由此看出，放置在不同文件夹下面的相同的图片，在相同像素密度下所取得的图片Bitmap大小是不同的，如上表格中，drawable-mdpi下的图片实际上是进行了缩放的。

程序得到的图片宽度 = 实际图片宽度 \* 设备像素密度 / 图片资源文件夹对应的像素密度

程序得到的图片高度 = 实际图片高度 \* 设备像素密度 / 图片资源文件夹对应的像素密度

由此可以看出如果图片放置在低密度文件夹中，而要在高像素密度设备上显示时，其会先进行放大，然后再显示，这样就会导致高像素密度设备上显示模糊。

注：图片Bitmap放大的过程可以在源码中找到，源码在BitmapFactory.decodeStream方法中。

## 第四章屏幕适配的解决办法

### 4.1、适配不同屏幕尺寸的解决办法。

本节内容为适配不同的屏幕尺寸解决办法

可以通过如下步骤去适应不同的屏幕尺寸:

* 确保您的布局可以充分调整大小以适应屏幕
* 根据屏幕配置提供合适的UI布局
* 确保正确的布局被施加到正确的屏幕
* 提供正确的缩放位图

#### 4.1.1、建议多使用 "wrap\_content" 和 "match\_parent"

为了去适配不同的屏幕尺寸建议多使用"wrap\_content" 和 "match\_parent"来定义布局文件上的各个组件的高和宽

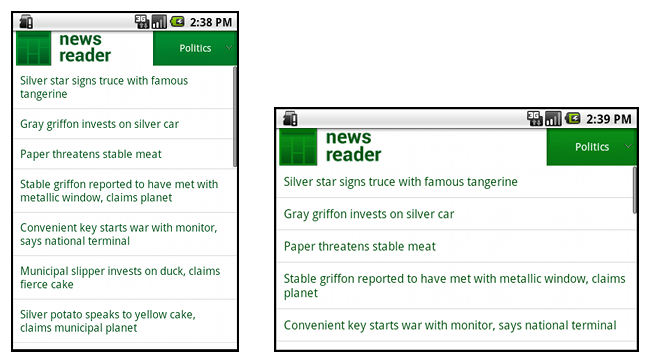
如果使用“WRAP\_CONTENT”来定义视图的宽度或高度，可以适应该视图中的内容的最小尺寸，而“match\_parent”（API8级之前也被称为“FILL\_PARENT”），使得组件扩展到匹配其父视图的大小。

通过使用"wrap\_content" and "match\_parent"来设定组件的尺寸而不用写死的代码，这使得你的view只需要所需的空间或者填充满所有可用的空间例如：

<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>

请注意该示例使用“WRAP\_CONTENT”和“match\_parent”的分量大小，而不是具体的尺寸。这使得布局能正确地适应不同的屏幕尺寸和横竖屏幕。

例如，这就是使用“WRAP\_CONTENT”和“match\_parent”这种布局看起来像在纵向和横向模式。但是请注意，该部件的尺寸的宽度和高度自动适应。



**Figure 1.** The News Reader sample app in portrait (left) and landscape (right).

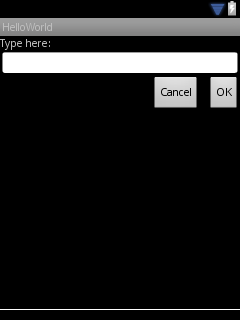
#### 4.1.2、建议多使用RelativeLayout（相对布局）

你可以使用 [LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) 布局方式和"wrap\_content" and "match\_parent" 进行组合编写出复杂的布局出来，但是[LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) 是很难精确控制它的子view之间的相对位置的。[LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html)所定义的布局都是线性的，假如你所需要的布局是非线性的布局方式[LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html)就很难控制了， [RelativeLayout](http://developer.android.com/reference/android/widget/RelativeLayout.html)就能很方便的解决这个问题吧，它允许我们灵活的布局各个组建的相对位置。例如，您可以在左侧布局一个子视图，另一个布局现对于在屏幕右侧。

For example:

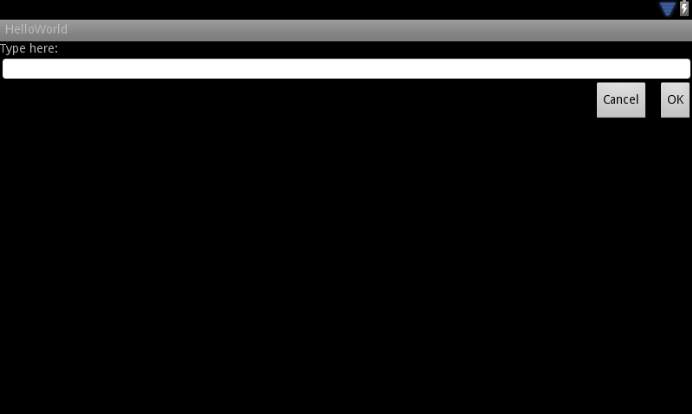
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent">  
    <TextView  
        android:id="@+id/label"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:text="Type here:"/>  
    <EditText  
        android:id="@+id/entry"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:layout\_below="@id/label"/>  
    <Button  
        android:id="@+id/ok"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_below="@id/entry"  
        android:layout\_alignParentRight="true"  
        android:layout\_marginLeft="10dp"  
        android:text="OK" />  
    <Button  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_toLeftOf="@id/ok"  
        android:layout\_alignTop="@id/ok"  
        android:text="Cancel" />  
</RelativeLayout>

Figure 2 shows how this layout appears on a QVGA screen.



**Figure 2.** Screenshot on a QVGA screen (small screen).

Figure 3 shows how it appears on a larger screen.



**Figure 3.** Screenshot on a WSVGA screen (large screen).

尽管组建的大小发生了变化，但是组建之间的相对位置并没有发生变化。

#### 4.1.3、建议使用尺寸标示符

这里只有这么多种方法，你可以从前面的章节中获得一个灵活的布局或相对布局。虽然这些布局由内部和周围的组件伸展空间适应不同的屏幕，但是它们可能不提供用于每个屏幕尺寸最佳的用户体验。因此，您的应用程序不应该只实现灵活的布局，也应该提供几种针对不同的屏幕配置可供选择的布局。通过使用配置限定符，允许运行时自动选择基于当前设备的配置相应的资源（例如一个不同的布局设计为不同的屏幕尺寸）。

例如，许多应用程序实现了“2”窗格模式为大屏幕（该应用可能显示的项目在一个窗格中的列表和在另一窗格中的内容）。平板电脑和电视是足够大的两个窗格同时适合在屏幕上，但手机屏幕必须单独显示它们。因此，要实现这些布局，你可以有以下文件

res/**layout**/main.xml, single-pane (default) layout:

<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-large**/main.xml, two-pane layout:

<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>

注意这个带有large标识符的布局文件，将自动的被大屏幕所选择（比如7英寸以上的的平板），剩下的（不带标识符）将自动的被小屏幕设备选择。

#### 4.1.4、建议使用最小宽度标识符

开发人员曾在Android3.2设备之前的困难之一是“大”屏幕尺寸问题，其中包括戴尔的Streak，最早的Galaxy Tab，一般7“平板电脑，但是许多应用程序可能需要显示不同的布局这一类中的不同设备（如5“和7”的设备），即使它们都被认为是“大”的屏幕。这就是为什么机器人推出的“最小宽度”限定符（主要）在机器人3.2。

该最小宽度限定符可以让你针对是在DP一定的情况下的最小宽度的屏幕。例如，典型的7“平板电脑拥有600 DP的最小宽度，所以如果你想你的UI会对这些屏幕两个窗格（但在小屏幕上一个列表），你可以使用两个相同的布局，从上一节单和双窗格的布局可以知道，在这里我们不适用大尺寸限定符，而使用sw600dp来表示两个窗格布局是屏幕上的最小宽度为600 DP：

* res/layout/main.xml, single-pane (default) layout:

<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-sw600dp/main.xml, two-pane layout:

<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>

这就意味着那些宽度大于和等于600dp的设备将自动的选择layout-sw600dp/main.xml，相反那些宽度小雨600dp的设备将自动的选择layout /main.xml此功能在Android3.2之前不适用

#### 4.1.5、建议使用布局别名

以上使用的最小标识符只能在Android3.2之后使用，因此你任然需要使用small, normal, large and xlarge标识符去适配早期版本，比如如果你想你的UI去适配手机、平板、TV等大尺寸设备时，你需要定义如下文件

* res/layout/main.xml: single-pane layout
* res/layout-large: multi-pane layout
* res/layout-sw600dp: multi-pane layout

最后两个文件夹由于只能在Android3.2之后使用，第一个能适用于所有版本设别。

为了避免这种重复同一个文件的平板电脑和电视，您可以使用别名文件。例如，您可以定义以下布局：

* res/layout/main.xml, single-pane layout
* res/layout/main\_twopanes.xml, two-pane layout

And add these two files:

* 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>

These latter two files have identical content, but they don’t actually define the layout. They merely set up mainto be an alias to main\_twopanes. Since these files have large and sw600dp selectors, they are applied to tablets and TVs regardless of Android version (pre-3.2 tablets and TVs match large, and post-3.2 will match sw600dp).

#### 4.1.6、建议使用横竖标识符

一些布局在横向和纵向方向运行良好，但大多可以从调整中受益。在新闻阅读器示例应用程序，下面介绍了如何布局的行为在每个屏幕的大小和方向

* **small screen, portrait:** single pane, with logo
* **small screen, landscape:** single pane, with logo
* **7" tablet, portrait:** single pane, with action bar
* **7" tablet, landscape:** dual pane, wide, with action bar
* **10" tablet, portrait:** dual pane, narrow, with action bar
* **10" tablet, landscape:** dual pane, wide, with action bar
* **TV, landscape:** dual pane, wide, with action bar

因此需要每个这些不同的设备定义不同的XML布局文件在res/layout/目录中。然后分配每个布局的各种屏幕配置，该应用使用了布局别名将它们匹配到每个配置

res/layout/**onepane.xml**:

<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**:

<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**:

<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**:

<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>

现在所有的可能的布局都被定义了，这是一个问题使用如何使用配置限定符映射正确的布局。您现在可以使用布局别名技术完成它

res/values/layouts.xml:

<resources>  
    <item name="main\_layout" type="layout">@layout/onepane\_with\_bar</item>  
    <bool name="has\_two\_panes">false</bool>  
</resources>

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

<resources>  
    <item name="main\_layout" type="layout">@layout/twopanes</item>  
    <bool name="has\_two\_panes">true</bool>  
</resources>

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

<resources>  
    <item name="main\_layout" type="layout">@layout/onepane</item>  
    <bool name="has\_two\_panes">false</bool>  
</resources>

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

<resources>  
    <item name="main\_layout" type="layout">@layout/twopanes</item>  
    <bool name="has\_two\_panes">true</bool>  
</resources>

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>

#### 4.1.7、建议使用.9.png图片资源

要想去适配不同的屏幕，不仅仅你的布局文件能适配各种屏幕，你的图片资源也要做到去适配不同的屏幕尺寸。例如一个按键背景图片就必须适配不管哪一种按键形状。

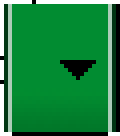
如果您的组件使用可以改变大小简单的图像，你很快就会发现这种解决方案不是很好的，因为运行时会拉伸或收缩均匀的图像。解决方案是使用.9.png，这是特殊格式PNG文件指示哪个区域可以拉伸哪些区域不能拉伸。

因此我们在做适用于可变尺寸的位图时建议使用.9.png图片资源。



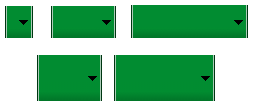
**Figure 4.** button.png

通过draw9patch工具你可以用左边和上面的点标注拉伸区域，用右边和下边的标记点去标记内容区。



**Figure 5.** button.9.png

注意黑色像素沿着边界。在顶部和左边框那些指明所述图像可拉伸的地方，并在右边和底部边界的那些指示其中内容应该放置。此外，注意.9.png扩展。必须使用该扩展，因为这是框架区别这是.9图像，而不是一个普通的PNG图像的区别。



**Figure 6.** A button using the button.9.png nine-patch in various sizes.

### 4.2、适配不同屏幕密度的解决办法

本节内容为适配不同的屏幕尺寸解决办法

这节课向您展示如何通过提供不同的资源和使用测量与分辨率无关的设备支持不同的屏幕密度。

#### 4.2.1、建议使用DIP

一个常见的错误是设计你的布局时你使用绝对像素来定义距离或尺寸。定义布局的尺寸与像素是一个问题，因为不同的屏幕有不同的像素密度，所以相同的像素数目可以对应于不同的设备上不同的物理尺寸。因此，指定尺寸时，一定要使用无论是DP或SP单元。一个DP是对应于一像素的实际尺寸，在160 dpi的密度无关的像素。 SP的是相同的基本单元，而是由用户首选的文本大小（它是一个独立的规模像素）缩放，所以你应该在定义文本大小（但从未进行布局大小）使用此计量单位。

For example, when you specify spacing between two views, usedp rather than px:

<Button android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="@string/clickme"  
    android:layout\_marginTop="20dp" />

When specifying text size, always use sp:

<TextView android:layout\_width="match\_parent"   
    android:layout\_height="wrap\_content"   
    android:textSize="20sp" />

#### 4.2.2、建议提供可供选择的位图

由于Android运行在多种屏幕密度的设备，你应该始终提供针对每个广义密度级别的位图资源：低，中，高，超高密度。这将帮助你实现所有的屏幕密度，良好的图形质量和性能。产生这些图像，你应该从你的矢量格式的原始资源，产生了使用以下尺寸的规模每次密度图像：

* xhdpi: 2.0
* hdpi: 1.5
* mdpi: 1.0 (baseline)
* ldpi: 0.75

这意味着，如果你为xhdpi设备生成一个200x200的图像，您应该生成150×150的资源图片为HDPI，100×100的MDPI，75x75的资源图片为LDPI设备。

图片放置方式如下:

MyProject/

res/

drawable-xhdpi/

awesomeimage.png

drawable-hdpi/

awesomeimage.png

drawable-mdpi/

awesomeimage.png

drawable-ldpi/

awesomeimage.png

Then, any time you reference @drawable/awesomeimage, the system selects the appropriate bitmap based on the screen's dpi.

### 4.3、适配特殊屏幕设备的解决办法

#### 4.3.1、建议声明该APP只能运行在手持设别上

由于该系统可以适配一般的大屏设配，但是你不能用大屏设备来适配你的程序。只要你遵循以上设计准这，你的应用程序应该在更大的屏幕也运行很好。但是，你可能会发现你的应用程序不是万能的，或也许你已经决定要发布应用程序的不同屏幕配置两个版本。在这种情况下，你可以使用<compatible-screens>元素来管理你的应用程序基于屏幕大小和密度的组合分布。外部服务，如谷歌播放使用这些信息来筛选适用于您的应用程序，使具有屏幕配置与声明只兼容设备可以下载你的应用程序。在<compatible-screens>元素必须包含一个或多个<screen>元素。每个<screen>元素指定的屏幕配置与您的应用程序是兼容的，同时使用了android：screenSizeand机器人：screenDensity属性。每个<screen>元素必须包括属性来指定一个单独的屏幕配置，如果任一属性丢失，那么元素是无效的（外部的服务，如谷歌播放会忽略它）。例如，如果您的应用程序，只有小型和正常大小的屏幕兼容，无论银幕密度，必须指定八个不同的<screen>元素，因为每个屏幕尺寸有四个密度配置。您必须声明，这些每一个;大小和密度的，你做notspecify任意组合被认为是屏幕配置与您的应用程序不兼容。下面是该清单条目看起来像如果你的应用是仅小，正常的屏幕尺寸兼容：

<manifest ... >  
    <compatible-screens>  
        <!-- all small size screens -->  
        <screen android:screenSize="small" android:screenDensity="ldpi" />  
        <screen android:screenSize="small" android:screenDensity="mdpi" />  
        <screen android:screenSize="small" android:screenDensity="hdpi" />  
       <screen android:screenSize="small"android:screenDensity="xhdpi" />  
        <!-- all normal size screens -->  
        <screen android:screenSize="normal" android:screenDensity="ldpi" />  
        <screen android:screenSize="normal" android:screenDensity="mdpi" />  
        <screen android:screenSize="normal" android:screenDensity="hdpi" />  
        <screen android:screenSize="normal" android:screenDensity="xhdpi" />  
    </compatible-screens>  
    ...  
    <application ... >  
        ...  
    <application>  
</manifest>

**Note:** Although you can also use the [<compatible-screens>](http://developer.android.com/guide/topics/manifest/compatible-screens-element.html) element for the reverse scenario (when your application is not compatible with smaller screens), it's easier if you instead use the [<supports-screens>](http://developer.android.com/guide/topics/manifest/supports-screens-element.html) as discussed in the next section, because it doesn't require you to specify each screen density your application supports.

#### 4.3.2、建议声明你的应用程序仅适用于平板设别

如果你不希望使用在手机上你的应用程序（也许您的应用程序真正使只能在大屏幕的意义上），或者你需要时间去优化它的小屏幕，可以防止小屏幕设备的使用下载你的应用程序在<supports-screens>清单元素。

例如，如果你希望你的应用程序只提供给平板电脑，你可以在你的清单如下声明元素:

<manifest ... >  
    <supports-screens android:smallScreens="false"  
                      android:normalScreens="false"  
                      android:largeScreens="true"  
                      android:xlargeScreens="true"  
                      android:requiresSmallestWidthDp="600" />  
    ...  
    <application ... >  
        ...  
    </application>  
</manifest>

它描述了两种不同的方式您的应用程序的屏幕尺寸支持：

•它宣称，该应用程序不支持屏幕尺寸的“小”和“正常”，这是历来没有片。

•它声明该应用程序需要的屏幕大小以最小使用面积为至少600dp宽。

第一种方法是运行的Android3.1或以上，因为这些设备申报尺寸基于广义屏幕尺寸的设备。该requiresSmallestWidthDp属性是用于运行Android3.2和更新的，它包括的能力为应用程序，以根据可用密度无关的像素的最小数目指定尺寸要求的设备。在这个例子中，应用程序声明的600dp最小宽度的要求，这通常意味着一个7“-or-更大画面。

你的尺寸的选择可能会不同，当然，根据了解你的设计作品在不同的屏幕尺寸;例如，如果你的设计作品以及只在屏幕上是9“或更大，您可能需要720dp的最小宽度。

美中不足的是，你必须对编译的Android3.2或更高版本的应用程序才能使用therequiresSmallestWidthDp属性。旧版本不明白这个属性，并会引发一个编译时错误。做最安全的事情是制定打击你已经设置forminSdkVersion的API等级相匹配的平台，您的应用程序。当你正在做最后的准备工作，以建立自己的候选版本，更改构建针对至Android3.2，并添加requiresSmallestWidthDp属性。超过3.2的Android版本只是忽略XML属性，所以这是一个运行时出现故障的风险。

关于为什么“最小宽度”的屏幕尺寸是支持不同的屏幕尺寸重要的更多信息，请阅读新工具管理的屏幕尺寸。

注意：如果你使用<支持屏>元素为反向方案（当你的应用程序不具有较大的屏幕兼容），并设置较大的屏幕尺寸属性为“假”，那么外部的服务，如谷歌播放不适用过滤。您的应用程序仍然会提供给更大的屏幕，但它运行的时候，它不会调整大小以适应屏幕。相反，系统会模拟手机的屏幕尺寸（约320dp X480dp;更多信息请参见屏幕兼容模式）。如果你想阻止你的应用程序被下载了大屏幕上，使用<兼容屏>，如上一节aboutDeclaring一个应用程序中讨论是仅适用于手机。

请记住，你要努力提供给尽可能多的设备尽可能您的应用程序通过应用所有必要的技术支持多个屏幕。你应该使用<兼容屏>或<支持屏>只有当你不能提供兼容于所有的屏幕配置，或者你已经决定提供不同版本的应用程序组不同的屏幕配置的。

#### 4.3.3、建议发布多个版本的APK适配不同设别

虽然我们建议您发布一个APK为您的应用程序，谷歌播放，您可以发布多个APK为同一个应用程序时，每个APK支持一组不同的屏幕配置（如清单文件中声明）。例如，如果要发布两种手机版本和应用程序的平板电脑版本，但你无法对两者的屏幕尺寸相同的APK工作，你其实可以出版两对的APK相同的应用程序清单。根据每个设备的屏幕配置上，谷歌播放将提供它，你已经宣布支持该设备的屏幕上的APK。

但是要小心，那发布多个APK为同一应用程序被认为是一种高级功能，大多数应用程序应该公布，可以支持多种设备配置只有一个APK。支持多种屏幕尺寸，尤其是在使用单个APK原因，只要您按照先导，以支持多个屏幕。

如果您需要了解如何在谷歌播放发布多个APK的更多信息，请阅读多APK支持。

## 第五章Android图标设计建议

### 5.1、Android Icon 多大时合适

这是一个简单的问题，但是即使是Android官方也只能提供较佳的建议文档，意思是没有固定的标准。

|  |  |  |
| --- | --- | --- |
| Android icons require five separate sizes for different screen pixel densities. Icons for lower resolution are created automatically from the baseline. | | |
| mdpi (Baseline): | 160 dpi | 1× |
| hdpi: | 240 dpi | 1.5× |
| xhdpi: | 320 dpi | 2× |
| xxhdpi: | 490 dpi | 3× |
| xxxhdpi: | 640 dpi | 4× |
|  | | |
|  |  |  |

### 5.2、Android Icon 尺寸

#### 5.2.1、Android launcher图标

Android launcher图标设计多大合适呢？

**答案是**: 48 px, 72 px, 96 px, 144 px, 192 px & 512 px (for Google Play Store).



48 × 48 (mdpi)  
72 × 72 (hdpi)  
96 × 96 (xhdpi)  
144 × 144 (xxhdpi)  
192 × 192 (xxxhdpi)  
512 × 512 (Google Play store).png

如果想要3D立体效果，建议制图时可以加一点角度。

#### 5.2.2、操作栏，对话框和图标选项卡

24 × 24 area in 32 × 32 (mdpi)  
36 × 36 area in 48 × 48 (hdpi)  
48 × 48 area in 64 × 64 (xhdpi)  
72 × 72 area in 96 × 96 (xxhdpi)  
96 × 96 area in 128 × 128 (xxxhdpi).png

这些图标用于在操作栏菜单。第一个数字是图标区域的大小，而第二个是文件大小。

#### 5.2.3、小的上下文 Icons

16 × 16 (mdpi)  
24 × 24 (hdpi)  
32 × 32 (xhdpi)  
48 × 48 (xxhdpi)  
64 × 64 (xxxhdpi)  
.png

小图标用于表面操作和/或用于特定项目提供状态。例如，在Gmail应用程序，每一个消息都有邮件标记为重要的一个星形图标。

#### 5.2.4、通知栏 icons

22 × 22 area in 24 × 24 (mdpi)  
33 × 33 area in 36 × 36 (hdpi)  
44 × 44 area in 48 × 48 (xhdpi)  
66 × 66 area in 72 × 72 (xxhdpi)  
88 × 88 area in 96 × 96 (xxxhdpi)  
.png

这些用于表示在状态栏应用的通知。他们应该是平的（无梯度），白色和正面的视角

### 5.3、Google Paly的Icon又该如何设计

Googleplay需要的图标在512×512像素的单额外版本.



Android Play Store Icon Size: 512 x 512 pixels

### 5.4、了解Android的图标大小

Android的态度对待自己的图标大小文档略有不同以其他平台，专注于不同的像素密度，而不是我们想要的简单的事实。然而我们略微考虑一下就知道了，他们为什么使用这种方法。

随着技术的进步，厂商纷纷拿出硬件与不断增加的分辨率和像素密度。在像素密度或每英寸点数（dpi）的增加，基本上意味着更多的图像信息可以与显示.

* MDPI – Medium Pixel Density – also known as the ‘Baseline’
* HDPI – High Pixel Density
* XHDPI – Extra High Pixel Density
* XXHDPI – Extra Extra High Pixel Density

**Note:** Android同样也支持LDPI屏幕的设配，但是你不需要特意的为他创建相应尺寸的资源，因为Android系统会将HDPI的资源缩小一半进行适配LDPI的设配。

### 5.5、Android用户界面图标大小

所有的Android图标遵循相同的比例规则，所以创建的图标文件时，你必须做出5版本，以涵盖所有平台上使用。下表显示的像素密度和图标大小的不同类型的所使用的Android的用户界面的图标：:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | MDPI (Baseline) | HDPI | XHDPI | XXHDPI | XXXHDPI |
| Scale | 1 x | 1.5 x | 2 x | 3 x | 4 x |
| DPI | ~ 160 dpi | ~ 240 dpi | ~ 320 dpi | ~ 480 dpi | ~ 640 dpi |
| App Launcher Icons | 48 px | 72 px | 96 px | 144 px | 192 px |
| Action bar Icons | 32 px (24px inset) | 48 px | 64 px | 96 px | 128 px |
| Small / Contextual Icons | 16 px (12px inset) | 24 px | 32 px | 48 px | 64 px |
| Notification Icons | 24 px (22px inset) | 36 px | 48 px | 72 px | 96 px |

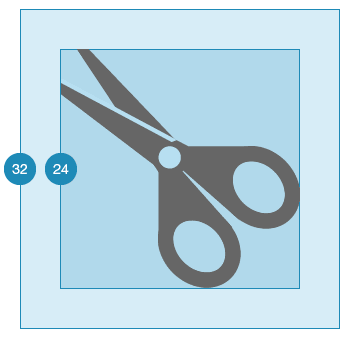
### 5.6、如何做出精美的Icon

专业定制设计的应用程序图标，在Play商店中脱颖而出，让你的应用程序备受关注.

[](http://www.creativefreedom.co.uk/icon-design-projects/app-icon-design/)

虽然用户界面图标遵循相同的比例规则应用程序启动器图标，他们也有自己的一些附加的显示规则:

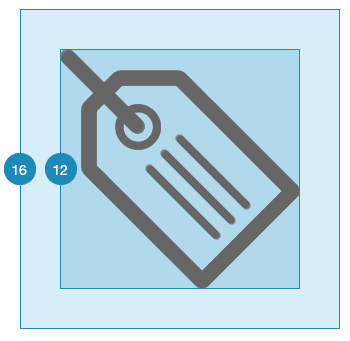
#### 5.6.1、Android操作栏图标大小



操作栏图标需要32×32像素，文件大小在基线尺寸（MDPI）。然而，图像本身必须是图标文件内居中不超过24×24个像素。

的图标设计必须是单一的颜色＃666666与透明度设置为60%

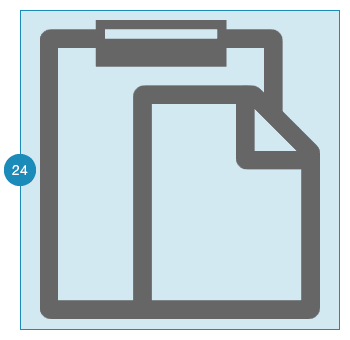
#### 5.6.2、Android小型/上下文图标大小



操作栏图标需要的16×16像素的大小的文件在基线尺寸（MDPI）。然而，图像本身必须是图标文件内居中不超过12×12个像素。

的图标设计必须是单一的颜色＃666666与透明度设置为60％。

#### 5.6.3、Android通知图标大小



操作栏图标需要24×24像素，文件大小在基线尺寸（MDPI）。然而，图像本身必须是图标文件内居中不大于22×22像素。

图标设计必须是单一的颜色白，#FFFFFF。

## 第六章Android屏幕适配建议综合

### 6.1、屏幕适配宗旨

**屏幕适配的宗旨：Android屏幕适配不只是让你的应用布局能正常显示，而且还要给用户带来精妙，舒适的布局享受。**

### 6.2、屏幕适配建议

1. 编写程序前应该首先定位运行的最低版本号，以及所支持的屏幕尺寸，以及是否支持竖屏，同时还要考虑到程序扩展性（特指对于以后更高分辨率手机出现后程序的适应性），同时需要该产品主体使用设备和设备版本。
2. 在具体写代码时不要使用px单位去定义布局；
3. 在具体写代码时使用dp单位去定义布局；
4. 在具体写代码时使用sp单位去定义字体的大小；
5. 建议在使用位图资源时首先考虑能否使用.9.png图片；
6. 建议在使用位图资源时需要考虑是否需要制作多张图，如果制作一张图是否能支持大多数手机的显示；
7. 尽量将点9图片放置在高像素密度资源文件夹中，这样即使在低像素密度手机上显示时会先对图片进行缩小再进行局部拉伸，但是在低像素密度手机上运行应用时，所有使用点9图片的地方都会对图片进行一次计算缩放，影响性能；
8. 要理解drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi、drawable-xxhdpi这几个图片资源的文件夹对应的像素密度为

|  |  |
| --- | --- |
| drawable-ldpi | 120dpi |
| drawable-mdpi | 160dpi |
| drawable-hdpi | 240dpi |
| drawable-xhdpi | 320dpi |

1. 同时需要考虑如果只放置一张图片资源时尽量制作高分辨率的图片；
2. 如果只放置一张图片资源时尽量制作高分辨率的图片的情况下需要考虑在底屏幕密度的设备上是否能完美的显示；
3. 要深入理解Android设备在加载图片资源时的顺序；
4. 要防止图片放置在低密度文件夹中，而要在高像素密度设备上显示时，其会先进行放大，然后再显示，这样就会导致高像素密度设备上显示模糊这种情况的发生；
5. 适配不同屏幕尺寸的解决办法建议多使用 "wrap\_content" 和 "match\_parent"；
6. 建议多使用RelativeLayout（相对布局）；
7. 对于特定屏幕尺寸建议使用尺寸标示符；
8. 对于支持平板设备的APP布局建议使用最小宽度标识符；
9. 对于支持平板设备的APP布局建议建议使用横竖标识符；
10. 适配不同屏幕密度的解决办法建议提供多种可供选择的位图；
11. 在屏幕适配的XML配置方面注意须有如下配置

<manifest ... >  
    <compatible-screens>  
        <!-- all small size screens -->  
 <screen android:screenSize="small" android:screenDensity="ldpi" />  
 <screen android:screenSize="small" android:screenDensity="mdpi" />  
 <screen android:screenSize="small" android:screenDensity="hdpi" />  
 <screen android:screenSize="small"android:screenDensity="xhdpi" />  
        <!-- all normal size screens -->  
 <screen android:screenSize="normal" android:screenDensity="ldpi" />  
 <screen android:screenSize="normal" android:screenDensity="mdpi" />  
 <screen android:screenSize="normal" android:screenDensity="hdpi" />  
 <screen android:screenSize="normal" android:screenDensity="xhdpi" />

。。。。。。。。等等配置  
    </compatible-screens>

1. 同时在作图时请参考如下表格标准：

|  |  |  |
| --- | --- | --- |
| Android icons require five separate sizes for different screen pixel densities. Icons for lower resolution are created automatically from the baseline. | | |
| mdpi (Baseline): | 160 dpi | 1× |
| hdpi: | 240 dpi | 1.5× |
| xhdpi: | 320 dpi | 2× |
| xxhdpi: | 490 dpi | 3× |
| xxxhdpi: | 640 dpi | 4× |
|  | | |
|  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | MDPI (Baseline) | HDPI | XHDPI | XXHDPI | XXXHDPI |
| Scale | 1 x | 1.5 x | 2 x | 3 x | 4 x |
| DPI | ~ 160 dpi | ~ 240 dpi | ~ 320 dpi | ~ 480 dpi | ~ 640 dpi |
| App Launcher Icons | 48 px | 72 px | 96 px | 144 px | 192 px |
| Action bar Icons | 32 px (24px inset) | 48 px | 64 px | 96 px | 128 px |
| Small / Contextual Icons | 16 px (12px inset) | 24 px | 32 px | 48 px | 64 px |
| Notification Icons | 24 px (22px inset) | 36 px | 48 px | 72 px | 96 px |