修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 说明 |
| V0.01 | 2010年12月3日 | 徐申龙 | 初始版本。 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[术语和概念 1](#_Toc279430715)

[Android 资源匹配 2](#_Toc279430716)

[aapt工具 6](#_Toc279430717)

[aapt使用简介 6](#_Toc279430718)

[更换 apk 包中的图片文件 6](#_Toc279430719)

[参考资料： 7](#_Toc279430720)

# 术语和概念

**屏幕大小**

实际物理尺寸，以屏的对角线长度衡量。为了简便，Android把各种屏幕尺寸分割为三个一般化尺度：大，普通，和小。应用程序可以给每个尺度分别提供布局—平台在实际屏幕上处理布局渲染是透明的。

**高宽比**

屏幕物理宽度对高度的比值。应用程序可以使用资源限定符long 和notlong，给特定的高宽比提供不同的布局资源。

**分辨率**

屏的像素总值。值得注意，尽管分辨率通常表述为*宽度x高度*，但没有隐含高块比。在Android平台，应用程序不能直接使用分辨率。

**密度**

基于屏的分辨率，指在屏幕物理宽度和高度上散布的像素。

一个低密度屏在宽度和高度散布可用像素较少的。而一个高密度屏有更多的--有时候明显多很多—像素分布在相同的区域。屏密度是重要的，因为在其他情况相同下，一个以屏幕像素定义高度和宽度的UI元素（像按钮）在低密度屏幕评将显得大一些，而在高密度屏幕上显得小一点。

简单起见，Android将各种实际的屏幕密度划分为三种一般的密度：高，中，和低。应用程序可以为这三种密度分别提供各自的资源—而平台会根据实际屏幕的密度放大或缩小资源。

**密度无关像素(dip)**

一种虚拟的像素单位，应用程序可以用来定义他们的UI，表述布局的尺寸或位置，而无须担心密度不同的问题。

密度无关像素相当于160dpi屏上的一个物理像素，基线密度由平台设定的。在运行时，平台会基于使用的屏实际密度透明地处理dip单位所需的任何缩放。dip单位与屏的像素转换很简单： 像素 = dips \* ( 密度 / 160 )。例如，在240dpi屏上，1dip就等于1.5物理像素。强烈推荐使用dip单位定义你的应用UI，因为这种方式可你保证你的UI在不同的屏上能正确显示。

# 多屏幕支持

Android能在各种各样的设备运行，这些设备屏的大小和分布各不相同。平台为应用在不同设备提供了一致的运行环境，为了应用的UI能适应屏幕而处理了很多复杂的问题。同时，平台也给应用程序开发者提供了可以精确控制UI的API，让他们处理特定屏幕尺寸和分别率上UI显示的问题。下面的内容讨论，如何让单独一个.apk文件部署到不同的设备时都能正确的显示。

## 屏支持的范围

从Android 1.6开始，平台可以支持多种不同大小和分辨率规格的屏，是平台能够在许多新类型和大小的设备上运行。如果在Android 1.6 或更新的平台上开发应用，适用Android平台兼容性特性来保证应用UI在所支持的各种不同的大小和分辨率正确渲染。

为了简化开发者为多种设备设计UI，让更多设备参于进来而不影响应用，平台把实际支持的屏幕尺寸和分辨范围作了划分：

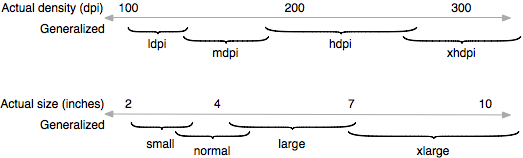
* 4种广义尺寸的集合：small(小)，normal (标准)，large (大)，和xlarge (超大)
* 4种广义密度的集合：ldpi(low)，mdpi(medium)，hdpi(high)，和xhdpi(extra high)

注意：xhdpi密度类别在Android 2.2(API级别8)才增加的。xlarge大小类别在Android 2.3(API级别9)增加的。

应用程序可以给四种广义尺寸提供自定义资源(主要布局)，以及四种广义密度提供各自资源(主要是绘图，如图片)。应用不需要关心屏幕实际物理尺寸或密度。在运行时，平台会为加载正确大小或密度资源处理相关事宜，从设备的屏的广义尺寸或密度出发，让他们适应屏幕的实际像素图。

广义尺寸/密度配置是围绕着基线配置，后者是指一个noraml大小和mdpi密度。为Android 1.5或更早编写的程序被认为是T-Mobile G1或类似设备所用的基线HVGA屏，这屏大小是normal而密度是mdpi。

每个广义屏幕配置跨越一定范围的实际屏幕密度和物理尺寸。例如，众多设备报告屏尺寸是normal的设备可能在实际尺寸和长宽比上有细微差别。类似的，各种hdpi密度屏的像素密度实际会有所区别。平台把这些差异抽象了，然而—应用程序只管为广义尺寸和密度提供UI设计 ，而让系统根据屏幕特性处理实际的UI渲染。



上图：阐析了Android平台上实际屏幕密度和尺寸与广义密度和尺寸配置之间的对应关系

尽管平台允许应用程序提供布局和资源给广义尺寸--密度配置，但是无须为这8个配置书写定制化的代码。平台提供了健壮的兼容特性，只要应用UI实现是正确的，它可以处理绝大多数应用在屏幕上的渲染工作。

## Android如何支持多种屏幕

Android支持多种屏幕的基础是一个内建的兼容特性集，他们共同管理使程序资源在屏幕上恰当渲染。平台处理其中大部分工作，但也提供了两个重要的方法来控制程序的显示，以供需要时使用：

* 平台提供了资源限定符集合，可以用来指导尺寸-和密度-限定的资源。尺寸资源限定符有small, normal,large,和xlarge。密度资源限定符有ldpi(低)，mdpi(标准)，hdpi(高)，和xhdpi(超高)。这些限定符的含义在上节已作说明。
* 平台也提供了<supports-screens>声明元素，它的属性有android:smallScreens, android:normalScreens, android:largeScreens, 和 android:xlargeScreens, 可以为让程序指定广义屏幕尺寸。另外，还有个属性android:anyDensity，它表示程序本身支持多种屏幕。

在运行时，平台为程序提供了三种类型的支持，以求最大可能在屏幕上显示：

1. 资源预缩放

平台会基于屏的密度从程序中自动装载特定大小或密度的资源，无缩放显示。假如没有匹配的资源，平台会加载默认的资源，并根据情况放大或缩小，以满足广义密度。平台认为默认的资源是为屏的基准密度”medium”(160)设计的。

例如，假如屏的密度是”high”，平台会加载有hdpi标签的资源，不用缩放的使用。如无此类资源，平台用默认的资源替代，把他们从基准密度(“medium”)放大到”high”。

1. 像素尺寸和坐标自动缩放

程序如声明不支持多种屏幕密度，平台会自动缩放任何程序使用绝对像素坐标，像素尺度值，和像素计算(指定视图的宽度或填充可能用到)。这样做是为了保证像素定义的屏幕元素显示时物理尺度，与基准密度(160)下基本相同的。对程序来讲，平台透明地处理这种缩放，并把缩放后的整个像素尺度告诉给应用，而非物理像素尺度。

例如，假如一个设备使用WVGA高密度平，即有480x800像素而大小与传统的HVGA屏几乎相同，有一个声明不支持多密度的程序在上面运行。这种情形，系统会向应用谎报其尺度为320x533。然后，当应用执行绘制操作，如要更新(10,10)到(100,100)矩形区域，系统同样自动转换，把他们缩小适当倍数，实际更新区域是(15,15) 到 (150, 150)。如果应用运行在低密度屏，坐标会缩小。

1. 大尺寸屏幕上兼容模式显示

如果屏幕尺寸大于程序所能支持的，根据 supports-screens元素指定的，平台依据基准尺寸(“normal”)和密度(“mdpi”)显示。对于大于基准的屏幕，只在整个屏幕的基准尺寸显示应用，其余部分显示黑色背景。

例如，假设一个设备用WVGA(480x800)中等密度的屏，被归为“大”号，但程序声明不支持大号屏；在此情况下，当程序查问屏尺寸时，系统又“撒谎”，告诉它自己只有320x480。此时，应用不会缩放，而是把它的320x480那部分界面像“邮票”一样贴在480x800的屏上。

一般来讲，这些兼容特性保证所有程序，包括那些为Android 1.5以及更早期版本编写的程序，都能在大多数屏幕上正确显示，尤其是对那些基准尺寸及更大尺寸的屏幕。

然而，注意为基准屏幕编写的程序可能需要一些调整才能在更小屏幕如QVGA上正确显示。

## 密度无关

密度无关是指程序在不同密度的屏上运行时，用户觉得他的界面大小看起来是一致。密度无关可适用于布局和图标等绘图资源。保持密度无关性是很重要的，因为在其他因素一致下，一个以像素定义长宽的UI元素(如按钮)在低密度屏上会变大而高密度屏上会变小。这种与密度相关的尺度改变会影响应用的布局、可用性以及与设备上其他应用外观的一致性。

平台默认为程序提供了密度无关性。他有三种处理方法：

* 通过预缩放绘制资源(在加载时缩放)
* 通过自动缩放布局中的密度无关像素(density-independent pixel , dp)值
* 通过自动缩放应用中使用的绝对像素值(仅仅当应用在其AndroidManifest.xml设置了android:anyDensity="false")

下面图片阐析了平台的密度无关性。可以看到布局和应用图标显示出来的大小是一致的，尽管屏大小、长宽比和密度都不相同。



图：在WVGA高密度(左边), HVGA中密度(中间), 和QVGA低密度(右边)表现的密度无关

大多数情况下，在应用中只需保证所有布局的尺度用密度无关像素(dp或dip)或者缩放无关像素(sip或sp，仅限文本)指定，就可以获得密度无关的好处。如果声明了[android:anyDensity="true"](http://androidappdocs.appspot.com/guide/practices/screens_support.html#attrs)，在使用绝对像素值时需要缩放。

## Manifest屏幕相关的属性

Android 1.6 引进了一个新的manifest元素，<supports-screens>，它可用控制应用不同类别的设备屏幕上显示。

|  |  |
| --- | --- |
| 属性 | 描述 |
| android:smallScreens | UI是否适合”small”屏—“true”表示可以，”false”则不可以。 |
| android:normalScreens | UI是否适合”normal”屏—“true”表示可以，”false”则不可以。默认是”true”。 |
| android:largeScreens | UI是否适合”large”屏—“true”表示可以，”false”则不可以。 |
| android:anyDensity | UI是否适合各种不同密度的屏-- “true”表示适合，”false”则不适合。   * 如果”ture”，平台关闭密度兼容特性，特别是绝对像素单位和计算的自动缩放，依靠程序使用密度无关像素单位去适应屏幕。 * 如果”false”，平台使能密度兼容特性。这种情况下，平台为程序提供缩放的虚拟屏幕像素映射，在它上布局和绘制UI就像是在一个中密度屏(160)上一样。平台根据实际屏幕密度透明地自动缩放程序像素单位值和计算值。 |

<supports-screens>各属性的默认值是不同的，主要取决于程序的manifest中属性 [android:minSdkVersion](http://www.gfan.com/dev/android/guide/topics/manifest/uses-sdk-element.html) 的值，还有android:targetSdkVersion的值，如果声明的话：

* 如果android:minSdkVersion 或 android:targetSdkVersion 是 "3" (Android 1.5) 或更低，除了android:normalScreens 外其他都是”false”。如果主要目标是Android 1.6平台，但也希望支持其他密度/屏幕尺寸，就需要把相应的值设为”true”。
* 如果android:minSdkVersion 或 android:targetSdkVersion 是 "4" (Android 1.6) 或更高，所有属性默认值都是”true”。如果程序需要Android 1.6的特性，但又不支持某些密度和/或尺寸，需要把相应属性设成”false”。
* 注意android:normalScreens默认值始终是true。

一般，一个屏幕尺寸相关属性(smallScreens, normalScreens, or largeScreens) 声明为”true”，就等于给平台一个信号，应用要自己管理各种屏幕尺寸的UI，不需要平台的大小兼容性(向虚拟HVGA显示区域)。而尺寸相关属性声明为”false”，则表明应用没有为屏幕尺寸作设计。不支持的尺寸可能造成的影响：

* 声明largeScreens=”false”，程序仍然能安装在大屏幕设备上。在大屏上运行，这个属性值会使平台以兼容模式运行程序，大屏留出基准区域(标准大小，中等密度)显示程序UI。
* 声明smallScreens=”false”，程序也能被用户安装在大屏幕设备上。但是，Android Market会在给用户的程序列表滤掉这些程序。实际上，这样阻止了用户在小屏设备上安装这类程序。

如果android::anyDensity属性声明为”true”，程序自己会处理在各种密度屏幕上的UI，使用实际的屏幕尺寸和像素。在这种情况，应用必须保证他使用密度无关像素声明UI尺度，用android.util.DisplayMetrics.density提供的缩放因子缩放实际像素值或计算。android::anyDensity属性的设置不会影响对绘制资源(像位图和nine-patch图片)预缩放， 这总是默认产生。

下面的例子展示了一个manifest，声明支持任意密度的大，中，小屏：

<manifest xmlns:android="http://schemas.android.com/apk/res/android">  
  <supports-screens  
          android:largeScreens="true"  
          android:normalScreens="true"  
          android:smallScreens="true"  
          android:resizable="true"  
          android:anyDensity="true" />  
  </manifest>

## 屏幕大小和密度相关的资源目录限定符

|  |  |  |
| --- | --- | --- |
| 屏幕特性 | 限定符 | 描述 |
| 尺寸 | small | 小屏的资源，像低密度QVGA。 |
| normal | 标准(基准配置)屏幕的资源，如 T-Mobile G1/HTC Magic。 |
| large | 大屏的资源。典型例子是平板电脑类设备。 |
| xlarge | 超大屏的资源。 |
| 密度 | ldpi | 低密度资源，适合100到140 dpi屏。 |
| mdpi | 中等密度资源，适用140到180 dpi屏。 |
| hdpi | 高密度资源，对于190到250 dpi屏。 |
| xhdpi | 超高密度资源。 |
| nodpi | 密度无关资源。平台不自动缩放带此标签的资源，不管屏的密度。 |
| 长宽比 | long | 适合长宽比明显高于(竖屏)和宽于(横屏) 标准的任意尺寸或密度的屏的资源。 |
| notlong | 适合长宽比接近标准配置的屏的资源。 |
| 平台版本 | v<api-level> | 资源仅限特定API级别或更高的。例如，应用设计在Android 1.5 (API 级别3)和Android 1.6(API级别4及更高)上运行，当程序在Android 1.5(API 级别3)运行时，任何用限定符–v4标记的资源都会被排除。 |

请注意密度和尺寸是两个独立参数，在系统里独立解析。例如，WVGA高密度看成一个标准屏，那是因为它的物理尺寸和T-Mobile G1差不多。另一方面，WVGA中密度屏被认为是一个大屏，--它的分辨率相同但像素密度要低，这就说它的物理尺寸要比基准屏大，想比之下能显示更多的信息。

这里有个例子，程序资源目录的结构，它支持低和高密度，采用不同的布局方案：

res/layout/my\_layout.xml //标准尺寸屏的布局

res/layout-small/my\_layout.xml // 小尺寸屏的布局

res/layout-large/my\_layout.xml // 大尺寸屏的布局

res/layout-large-land/my\_layout.xml //大尺寸竖屏布局

res/drawable-ldpi/my\_icon.png //低密度图标

res/drawable-mdpi/dpi/my\_icon.png // 中密度图标

res/drawable-hdpi/my\_icon.png // 高密度图标

res/drawable-nodpi/composite.xml // 密度无关的图标

# Android 资源匹配

为了包含备用资源，可以创建相似的资源文件夹，在文件夹名后附加限定词来说明适用的配置（语言，屏的方向，等等）。比如，这里有个工程有一个英文的字符串资源文件，另一个是法文的：

MyApp/

res/

values-en/

strings.xml

values-fr/

strings.xml

Android支持一些修饰类型，每个都有各自可变的值。可以把它们加到资源文件夹名字的后面，以破折号连接。你可以增添多个限定词在文件夹名里，但是他们必须按一定的次序排列。比如，一个包含绘制资源文件夹名在全部配置时是这样的：

MyApp/

res/

drawable-en-rUS-port-160dpi-finger-keysexposed-qwerty-dpad-480x320/

一般的，你为资源会只指定一些特定的配置选项，你可以从完整列表中删掉任何的值，只要保留的值次序不变。

MyApp/

res/

drawable-en-rUS-finger/

drawable-port/

drawable-port-160dpi/

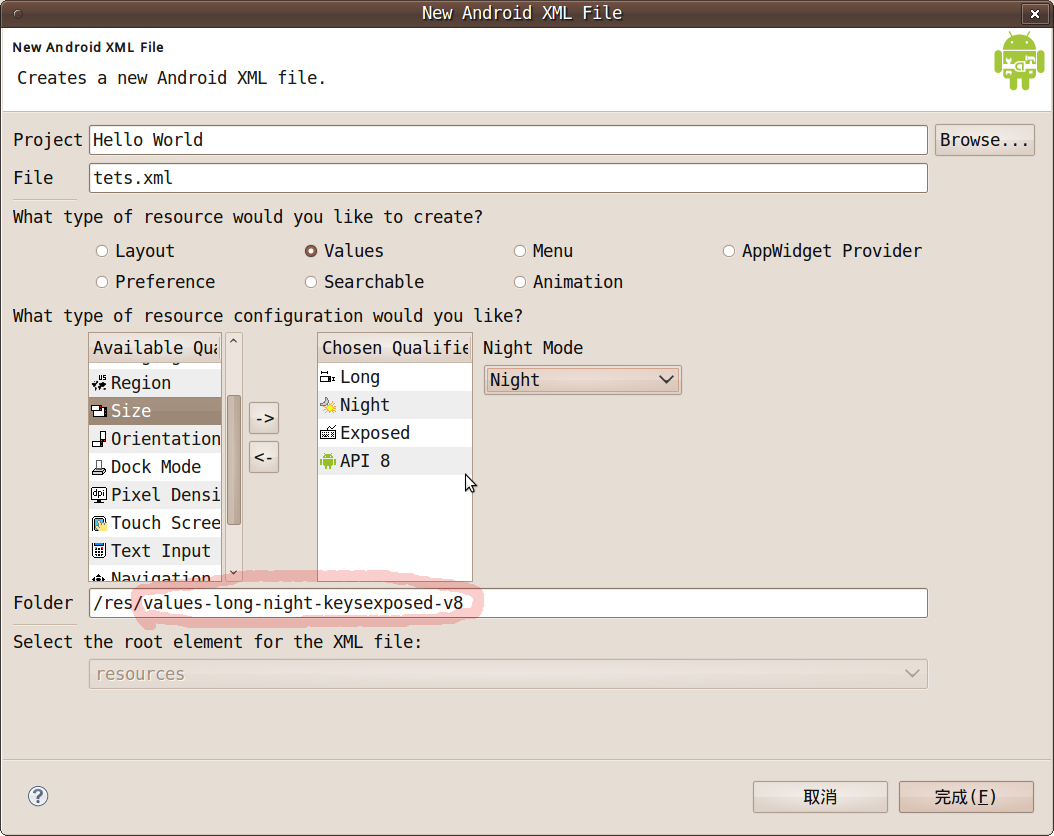
drawable-qwerty/

|  |  |  |
| --- | --- | --- |
| 限定词 | 取值 | 含义 |
| 语言（Language） | en, fr, es等 | [ISO 639-1](http://www.loc.gov/standards/iso639-2/php/code_list.php)规定的两个小写字母的语言代码。 |
| 地区（Region） | rUS, rFR, rES等 | [ISO 3166-1-alpha-2](http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html)规定的两个大写字母国家代码，并以”r”为前缀。 |
| 屏方向（Screen Orientation） | land | 横屏（Landscape ） |
| port | 竖屏（Portrait） |
| square | 方屏（Square） |
| 屏密度（Pixel Density） | xhdpi | eXtend High Density 超高密度 |
| hdpi | High Density 高密度 |
| mdpi | Medium Density 一般密度 |
| ldpi | Low Density 低分别率 |
| 触屏类型（Screen Touch） | notouch | No touch |
| stylus | Stylus 触笔，一般是指电阻屏。 |
| finger | Finger，指触，一般适用电容屏。 |
| 屏的大小（Screen Dimension） | 320x240，640x480等 | 屏幕宽高像素值，值大的写在前面。 |

资源目录名限定规则：

* 值用破折号分割。
* 值大小写敏感的。一个横屏的drawable目录必须命名为drawable-port，不能命名为 drawable-PORT。不能有名为drawable-port 和 drawable-PORT两个目录，即使是打算用"port" 和"PORT"引用不同参数值。
* 每种限定词类型只有一个值(这就是说不能指定目录drawable-rEN-rFR/)。
* 可以为特定的配置指定多个参数，但是他们必须按照一定的次序。如， drawable-en-rUS-land 不适用竖屏的US-English 设备。
* 资源可以有多个限定的目录，Android会为当前配置找出最匹配的目录。
* 所有目录，不管是否限定的，都存在res/ 目录下。限定的目录不能嵌套（不允许有目录res/drawable/drawable-en）。

Google提供了一个ADT的Eclipse插件，在Eclipse新建XML文件时它提供了资源配置的功能，我们只需选择配置项及值，以上的规则工具会自动处理了，如下图所示：



Android 在运行时根据设备的配置从各种资源选出优先的文件，选择步骤如下：

1. **排除掉配置与设备不匹配的资源。如，屏的像素密度是108dpi，那么就应该排除MyApp/res/drawable-port-92dpi/。**

MyApp/res/drawable/myimage.png

MyApp/res/drawable-en/myimage.png

MyApp/res/drawable-port/myimage.png

~~MyApp/res/drawable-port-92dpi/myimage.png~~

**2．选取匹配配置最多的资源。比方对于本地化为en-GB横屏设备，有两个匹配配置候选项：MyApp/res/drawable-en/myimage.png与MyApp/res/drawable-port/myimage.png。目录MyApp/res/drawable要排除，因为它没有任何匹配的配置，而其余的都有一个匹配的配置。**

~~MyApp/res/drawable/myimage.png~~

MyApp/res/drawable-en/myimage.png

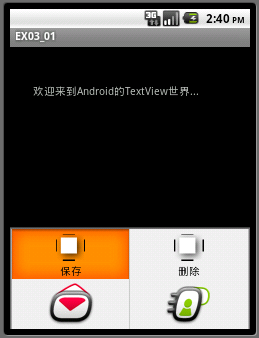
MyApp/res/drawable-port/myimage.png

**3．根据配置的优先级选取最终匹配的文件，即列在上面表格的参数顺序。那就是说，匹配语言比匹配方向要重要，所以我们最终选择与特定语言相关的文件，MyApp/res/drawable-en。**

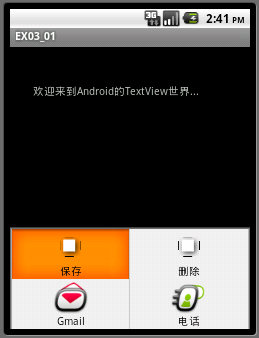
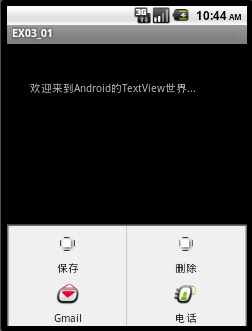
MyApp/res/drawable-en/myimage.png

~~MyApp/res/drawable-port/myimage.png~~

我把几张图片分别放入不同限定目录下，在模拟器上运行的效果如下：

res/drawable-72dpi/ res/drawable-120dpi/

res/drawable/ res/drawable-240dpi/

模拟器设置的密度是160dpi，当图片放入目录密度限定小于160dpi时，被放大了；大于160dpi时，图片被缩小了。这是一个线性关系，设备的分别率与资源限定密度的比值。所以，密度小的限定目录下，应该放置小的图片；而限定密度大的目录下放置大的图片。这样，应用运行在密度小的设备时，就会使用小图；在密度大的设备时，就会适用大图；就不会出现图片被过分放大或缩小的现象。

# 屏幕适应的最佳实践

多屏幕适应的目标是借助平台创建能在任意显示屏上正常运行，对任意一般化的屏幕配置能正确执行的应用程序。

保证应用程序在不同屏上正确显示是容易。这里有张简单的清单：

1. 在XML布局文件中优先使用wrap\_content, fill\_parent和dip单位。
2. 避免使用 AbsoluteLayout
3. 在代码中不使用硬编码的像素值
4. 使用密度和/或分辨率指定资源
5. **在XML布局文件中优先使用wrap\_content, fill\_parent和dip单位**

当定义在XML布局文件中定义视图的 layout\_width和layout\_height时，使用 wrap\_content, fill\_parent 或 dip会保证在当前设备的屏幕上视图能获得一个合适的大小。例如，一个以layout\_width="100dip"表述的视图在一个HVGA@160密度显示屏上会有100个像素宽，而在WVGA@240显示屏上却有150个像素宽，但是视图占据的物理空间是差不多的。

类似，推荐使用sp（scale-independent pixel,用户设置的缩放因子）或dip（如果不允许用户缩放文本）定义字体大小。

1. **避免使用 AbsoluteLayout**

AbsoluteLayout 是Android UI 工具包提供的一个布局容器。然而与别的布局不同，AbsoluteLayout强制使用固定的位置，这可能容易导致用户界面在不同显示屏上工作不正常。正因为如此，在 Android 1.5 （API 级别3）不推荐使用。

要实现相同的布局，用使用 FrameLayout替代，并设置孩子的layout\_margin属性。这种方法更灵活，在不同屏上能得到更好的效果。

1. **在代码中不使用硬编码的像素值**

为了性能原因和保持代码简洁，Android框架API以像素作为描述尺度或坐标值的标准单元。这意味着视图的尺度在代码中总是以像素表述。例如，假如myView.getWidth()返回10，这个视图就是10像素宽。在某些情况，你可能需要在代码中缩放像素值。下面提供更多这方面内容。

**从dp转换到像素**

在一些情况，需要用dp描述尺度，然后转化到像素。设想一个应用要在用户手指移动至少16个像素后就要识别一个滚动手势。在一个基准屏，用户在识别手势前要移动手指 16像素/160dpi=1/10英寸(或2.5mm)。在高(240)密度屏的设备上，用户只需移动手指16像素/240dpi = 1/15英寸(或1.7mm)。距离短多了，程序因此显得更灵敏了。为解决此问题，手势阀值在代码中必须用dp表示，再转成实际的像素值。

// The gesture threshold expressed in dp

private static final float GESTURE\_THRESHOLD\_DP = 16.0f;

// Convert the dps to pixels

final float scale = getContext().getResources().getDisplayMetrics().density;

mGestureThreshold = (int) (GESTURE\_THRESHOLD\_DP \* scale + 0.5f);

// Use mGestureThreshold as a distance in pixels

根据屏密度把dp转换成像素时必须使用android.util.DisplayMetrics.density域指定的缩放因子。通过Context或Activity获取屏度量。在中等(160)密度屏，DisplayMetrics.density等于”1.0”。详细情况参见类DisplayMetrics文档。

**使用预缩放配置值**

ViewConfiguration类可以存取Android框架中最常见的距离、速度，和时间。例，可按如下方法获得框架中使用的滚动阀值的像素距离：

ViewConfiguration.get(aContext).getScaledTouchSlop()

无论屏密度，以前缀getScaled开始的方法保证返回一个可以正常显示的像素值。

1. **使用密度和/或分辨率指定资源**

即使平台提供了大小和密度兼容的特性，在特定屏幕大小或密度屏上，还是有可能希望调整应用的UI。提供大小或密度相关的资源—资源包，布局，字符串，等等。如果需要，也可以控制图片资源的缩放。下面的章节作详细讨论。

**定制资源和目录**

如果需要精确控制程序在各种屏容貌，简单地调整特定配置资源目录下的资源包和布局。例如，考虑在中密度和高密度屏上显示一个图标。创建两个不同大小的图标(如，100x100给中密度的，150x150给高密度的)，并放入两个相应修饰的目录：

res/drawable-mdpi/icon.png   // for medium-density screens

res/drawable-hdpi/icon.png   // for high-density screens

假如一个资源目录名没有密度修饰符，平台就会假设此目录的资源是给作为基线的中密度设计的。不推荐把图片等和密度相关的资源放在默认目录中。

**位图和nine-patch图片的预缩放和自动缩放**

当从程序资源中加载位图或nine-patch图片，平台会根据屏幕的密度把图片预缩放。如，一张在re/drawable/目录里100x100位图显示在高密度屏时，Android会自动把图标放大成150x150位图。

预缩放机制与图片来源无关。例如，定位于高密度屏的应用可能只有res/drawable-hdpi/目录下有位图。如果程序在中密度屏上运行，240x240的图标就会缩小为160x160。

平台根据需要预缩放资源，而不管程序是否密度兼容(由supports-screens元素的android:anyDensity属性指定)。然而，在密度兼容下运行，只要预缩放的位图和其他资源被装入中密度屏，平台会不断报告其大小。比如，当密度兼容时，把默认资源中76x76图像显示在高密度屏上，平台内部会预放大图片至114x114。但是，API仍然报告图片大小为76x76。假如程序不注意直接操作缩放的图片，这种差异会导致不可预料的行为，但是可以认为这是为了尽可能保持既有程序性能的合理折中。

程序内部如果创建内存位图，在其上绘制一些东西，然后显示在屏上，这种情况下上面的规则就不适用。平台在绘制时才自动缩放这种位图。这种案例的其他副作用是绘制在这样位图上的字体会随之缩放，当位图最终渲染在屏上，会出现缩放的痕迹。

有些情况不需要Android自动缩放一个资源。最简单的方法是他放在”nodpi”资源目录：

res/drawable-nodpi/icon.png

类BitmapFactory.Options也可以控制缩放，他可以设定是否预缩放位图和位图的密度。例如，如果从web服务器加载位图，可能希望位图的密度要高。当禁用预缩放，最终的位图是在自动缩放模式。设置密度位图(通过BitmapFactory.Options指定或撤销)可以在运行时缩放。

自动缩放比预缩放要更占用CPU，但是使用的内存较少。有关自动缩放更多的信息可以参考类BitmapFactory, Bitmap, 和Canvas的文档。

# aapt工具

aapt是英文Android Asset Packaging Tool的缩写，这是一个创建、查看、更新 zip 格式兼容的文档（如以.zip, .jar, .apk为扩展名的文件）的工具，例如把Android程序的各种资源文件（如.xml文件，png格式图片，以及文本等）编译成二进制资源包，并把相应的索引写入到R.java 和 Manifest.java，供Android代码使用。

因为编译程序需要aapt，Android SDK都自带了appt，如platforms/android-7/tools/aapt，而不是放在SDK中的顶层tools目录中， 可见aapt跟版本有关系，不同版本的aapt之间可能不兼容。如果你编译了Android，那么在目录out/host/linux-x86/bin/中就可以找到aapt。

## aapt使用简介

下面介绍aapt一些常用的功能，让我们实际了解aapt的作用。更多的相关信息，可以看appt的帮助文档。

1．打包

$out/host/linux-x86/bin/aapt package -M packages/apps/SpeechRecorder/AndroidManifest.xml -S packages/apps/SpeechRecorder/res -F out/target/product/generic/obj/APPS/SpeechRecorder\_intermediates/package.apk

这是一个aapt打包命令（package），打包时必须指明manifest文件，资源路径，输出的路径和文件名。这里只是把AndroidManifest.xml以及图片、字符串等资源打包，在编译Android应用包后续过程中还会把编译的.dex格式的程序加入到包中，并对齐、签名。

2．删除包文件

$aapt r out/target/product/generic/system/app/Launcher2.apk resources.arsc

删除程序包Launcher2.apk中名为resources.arsc的文件。resources.arsc是res目录下各个子目录strings.xml编译后的文本资源，包括各种语言的翻译。

3．显示包的信息

$aapt list -v out/target/product/generic/system/app/Launcher2.apk

list子命令可以显示压缩包里的文件名，-v选项可以让aapt显示各个文件大小，CRC-32以及压缩方法、压缩率等详细信息。

4．转储包的内容

$aapt dump xmltree Calculator.apk AndroidManifest.xml

转储Calculator.apk包中的XML文件 AndroidManifest.xml。

## 更换 apk 包中的图片文件

更换图片分三步：删除原来的图片，准备替换的文件，添加新的图片。这里要注意的是文件的路径，假如要替换包drawable-hdpi下的图片advanced.png，那新的图片存放的路径必须是res/drawable-hdpi/advanced.png，而包文件如Calculator.apk必须与目录res在同一个路径下：

~$ aapt r Calculator.apk res/drawable-hdpi/advanced.png

~$ mv icon\_bullet\_key.png res/drawable-hdpi/advanced.png

~$ aapt a Calculator.apk res/drawable-hdpi/advanced.png

当然往apk包中加入新的图片更简单，只需要后面两步即可。aapt会把res下图片按其存放相对路径放入压缩文件中，所以一次可以加或更新多张图片。

# 参考资料

1. Supporting Multiple Screens

Android 官方文档，讨论了Android平台在支持不同屏幕时的策略，以及我们在编写多屏幕适应屏幕的应用程序时的技巧，包含在SDK开发文档，一个在线SDK文档的网址：

<http://androidappdocs.appspot.com/guide/practices/screens_support.html>

1. Android UI design reference

一个很好的设计Android UI工具。你可以提供图片，输入文字，或者选择剪贴画，然后调节一些参数，这个工具就能生成一阵套图片，分别为大/中/小密度，选中/没选中，以及不同版本的SDK生成文件。

http://code.google.com/p/android-ui-utils/

http://android-ui-utils.googlecode.com/hg/asset-studio/dist/icons-launcher.html

http://android-ui-utils.googlecode.com/hg/asset-studio/dist/icons-tab.html

http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

1. DroidDraw

Android平台用户界面设计器/编辑器，有Linux，Windwos，Mac OS X平台的独立的可执行程序，网址是：<http://www.droiddraw.org/>

1. User Interface Guidelines

指导美工设计图标和桌面小饰件(Widget)。教导程序员设计好Activity的行为和菜单命令安排，使程序有良好的用户交互体验。

<http://androidappdocs.appspot.com/guide/practices/ui_guidelines/index.html>

1. Application Resources

Android 应用可以使用的资源类型，存放位置，为特定的设备设置创建可选的资源；

如何使用资源，在程序代码里或其他XML资源里；在活动(Actitvity)运行时管理配置的修改；字符串的本地化；以及各种资源类型的参考手册，描述他们的XML元素、属性和语法。

<http://androidappdocs.appspot.com/guide/topics/resources/index.html>

1. User Interface

布局声明，常见的布局；菜单、对话框创建；UI事件处理，用户通知；样式和主题；以及视图如何绘制，等等用户界面相关的主题。

<http://androidappdocs.appspot.com/guide/topics/ui/index.html>