# 屏幕适配

Android运行在提供不同屏幕尺寸和密度的多种设备上。 对应用程序而言，Android系统提供一致的跨设备的开发环境并且处理适配不同显示屏幕的大部分工作。同时，系统提供API，允许针对不同的屏幕尺寸和密度来控制的应用程序UI，从而为不同的屏幕配置来优化UI设计。

尽管系统会进行缩放和调整，以使应用程序在不同的屏幕上运行，仍然应该尽量为不同的屏幕尺寸和密度来优化应用。最大限度的为所有设备优化用户体验，这样用户才会认为应用程序是真正的为他们的设备设计的，而不是简单的拉伸或缩放来适应他们的设备。

按照本文档中所述的做法，可以创建一个应用程序，使之在所有支持的屏幕配置上正确显示，并使用单一的.apk文件提供优化的用户体验。

注：本文档中的信息假设应用程序是为Andr​​oid 1.6（API等级4）或更高版本设计。如果要应用程序支持Android 1.5或更低，请首先阅读 [Strategies for Android 1.5](http://developer.android.com/guide/practices/screens-support-1.5.html).   
  
同时，要注意的Andr​​oid 3.2推出新的API，更精确地控制应用程序使用不同屏幕尺寸的布局资源。如果你正在开发为平板优化的应用程序，这些新功能尤其重要。有关详情，请参阅 [为Android 3.2设计平板布局](#平板电脑设计布局)

**屏幕支持概述**

本节概述Android对多个屏幕的支持，其中包括：在本文中和API中会用到的术语和概念的介绍，系统支持的屏幕配置的总括，以及API和底层屏幕适配特性的概览。

**术语和概念**

*屏幕尺寸*

实际的物理尺寸，以屏幕的对角线为准。

为简单起见，将所有的实际尺寸分为四个广义的尺寸：small（小）, normal（正常）, large（大）, extra large（特大）。

*屏幕密度*

屏幕的物理面积内的像素数量，通常指的DPI（每英寸点数）。例如，“低”的密度屏幕比“正常”或“高”密度屏幕在一个给定的物理面积内具有较少的像素。

为简单起见，将所有的实际密度分为四个广义的密度： low（低）, medium（中等）, high（高）, extra high（超高）。

*方向*

从用户的角度来看，屏幕的方向。是横向或纵向，也就是说，屏幕的比例是高或者宽。需要注意的是，不仅要在不同的屏幕方向做不同的操作，还要考虑到用户在运行时通过转动设备切换屏幕方向的情况。

*分辨率*

在屏幕上的物理像素总数。在支持多个屏幕时，应用程序不直接与分辨率相关;应用程序应该只与屏幕大小和密度相关。

*密度无关的像素（DP）*

在定义UI布局时应该使用的虚拟像素单元，它用一种密度无关的方式来表达布局尺寸或位置。

在160 dpi的屏幕上的一个物理像素，是“中等”的密度屏幕系统所承担的基准密度。系统在运行时透明地处理任何DP单位，必要时根据实际使用的屏幕密度缩放。 DP单位转换为屏幕像素是简单的：PX = DP \*（DPI / 160）。例如，在240 dpi屏幕，1 DP等于1.5物理像素。定义应用程序的用户界面时，应该总是使用DP单位，以确保不同密度的屏幕上正确显示UI。

**屏幕支持范围**

从Android 1.6（API Level 4）开始，Android提供了对多个屏幕尺寸和密度的支持，以反映出设备可能有的不同的屏幕配置。可以使用Android系统的功能，为每个屏幕配置优化应用程序的用户界面，从而确保应用程序为每个屏幕提供正常并且尽可能最佳的用户体验。

为了简化为多种屏幕设计用户界面，Android划分了实际的屏幕尺寸和密度范围：

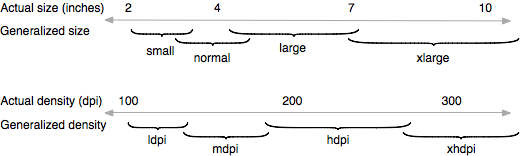
* 广义的尺寸大小集合： *small（小）*, *normal（正常）*, *large（大）*, and *xlarge（超大）*

**注**：从Android 3.2（API Level 13）开始，这种尺寸集合被废弃，取而代之的是一种基于可用屏幕宽度来管理屏幕尺寸的新技术。所以针对Android 3.2以及以上版本的开发，可以参考 [为Android 3.2设计平板布局](#平板电脑设计布局)来获取更多信息。

* 广义的密度集合： *ldpi* (低), *mdpi* (中), *hdpi* (高), and *xhdpi* (超高)

广义的大小和密度都围绕一个基准配置，即正常的大小和MDPI（中）密度。这个基准配置即第一款Android设备的配置，此设备即拥有一个HVGA屏幕的T-Mobile G1（直到Android 1.6，这是Android支持的唯一的屏幕配置）。

每个广义的大小和密度跨越了一系列实际的屏幕尺寸和密度。例如，两个设备，这两个设备报告的正常屏幕尺寸和宽高比可能与手工测量时略有不同。同样，两个设备报告的hdpi的屏幕密度可能与真正的像素密度略有不同。 Android抽象了应用程序的这些差异，因此可以提供广义的大小和密度设计的用户界面，让系统需要处理任何最终的调整。图1显示了不同的大小和密度大致分类成不同的大小和密度组。



**图1.** 解释了Android对实际的尺寸和密度到广义的尺寸和密度（数字是不准确的）的大致映射。

每一个UI设计需要一个最小的空间，每一个上文提到的广义的屏幕大小都拥有一个由系统定义的最小分辨率。这些空间是以“dp”为单位的，当定义布局时，也应当使用相同的单位，它使系统避免了对屏幕密度变化的顾虑。

* *xlarge* 屏幕至少 960dp x 720dp
* *large* 屏幕至少 640dp x 480dp
* *normal* 屏幕至少 470dp x 320dp
* *small* 屏幕至少 426dp x 320dp

为不同屏幕尺寸和密度优化应用程序的用户界面，可以提供任何广义的大小和密度的[alternative resources](http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources)（选择性资源）。通常情况下，应该为不同的屏幕尺寸提供alternative layouts并且为不同的屏幕密度提供alternative bitmap images。在运行时，系统基于当前设备的广义屏幕尺寸或密度为应用程序采用适当的资源，

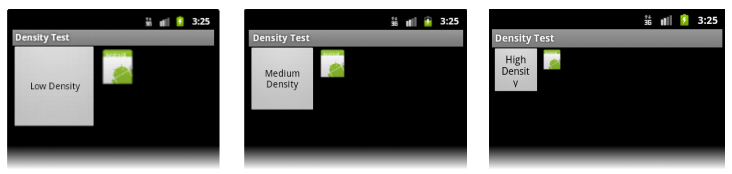
不需要为每一个屏幕大小和密度的组合提供选择性资源。系统提供了强大的兼容特性，可以处理在任何设备的屏幕上呈现应用程序的大部分工作，开发者只要实现UI技术，允许它正常调整。

**注:** 定义广义屏设备的大小和密度的特点是相互独立的。例如，WVGA的高密度的屏幕被认为是一个正常大小的屏幕，因为它的物理尺寸是相同的T-Mobile G1（Android的设备和基线的屏幕配置）。另一方面，WVGA的屏幕中密度被认为是一个大尺寸屏幕。虽然它提供了相同的分辨率（像素数相同），WVGA的屏幕中密度较低的屏幕密度，这意味着每个像素的物理面积较大，因此，整个屏幕是比基准（正常大小）的屏幕更大。

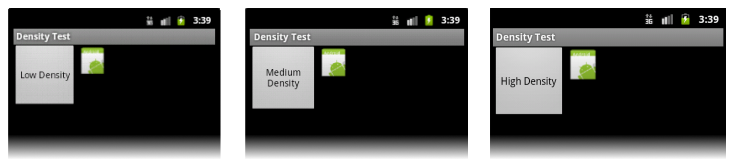
**密度的独立性**

当应用程序在具有不同密度的屏幕上显示时，它保留了用户界面元素（从用户的角度来看）的物理尺寸，即实现了“密度独立”。

保持密度的独立性是非常重要的，因为，没有它，一个UI元素（如按钮），在低密度屏幕上显示较大而在高密度屏幕上显示较小。这样的密度相关的大小的变化可能会导致应用程序的布局和可用性问题。图2和图3显示了一个应用程序不提供密度独立性时的差异。



**图 2.** 应用程序不支持密度独立，在低，中，高密度屏幕上显示。



**图 3.** 应用程序支持密度独立，在低，中，高密度屏幕上显示，支持良好

Android系统可以通过两种方式使应用程序实现密度独立：

* 系统缩放dp单位以适应当前屏幕密度
* 系统基于当前屏幕密度按需要缩放可曳资源到合适尺寸

在图2中，text view和bitmap以像素为单位指定尺寸（像素单位），所以视图的物理尺寸在地密度屏幕上要大一些而在高密度屏幕上要小一些。这是因为实际的屏幕尺寸虽然可能是相同的，具有高密度的屏幕每英寸的像素越多（相同数量的像素适合在一个较小的区域）。

在图3中，通过密度无关的像素（DP单位）指定布局尺寸。因为密度无关的像素的基线是一个中等密度的屏幕，一个中密度屏幕的设备看起来和图2中是一样的。然而，对低密度和高密度的屏幕，系统按比例调高或调低密度无关的像素值，分别以适应屏幕。

在大多数情况下，可以简单的通过用密度无关像素（dp units）来定义所有的布局尺寸或者适当的通过"wrap\_content"来确保应用程序密度无关。然后系统依据当前屏幕密度的比例因子按比例决定bitmap drawable以显示合适的尺寸。

然而，位图缩放可能会导致模糊或像素化的位图。为了避免这些人为效果，应该为不同密度提供选择性位图资源。例如，应该为高密度的屏幕提供更高分辨率的位图从而系统可以不必去调整为低密度屏幕准备的位图。以下部分描述如何为不同的屏幕配置提供选择性资源。

**如何支持多屏幕**

Android可以为当前的屏幕配置通过适当的方式管理应用程序的layout和bitmap drawables的表现，这种能力是Android支持多屏幕的基础。系统处理的应用程序屏幕适配的大部分工作，尽管如此，为了更妥善处理不同的屏幕配置，应该注意以下几点：

* **在清单文件中明确声明应用程序支持的所有屏幕尺寸**

通过声明应用程序支持的屏幕尺寸，可以保证只有那些屏幕尺寸被应用程序支持的设备才可以下载该应用程序。声明对不同屏幕尺寸的支持，也可以影响系统如何在更大的屏幕展现应用程序—特别地，应用程序是否在[屏幕兼容模式](http://developer.android.com/guide/practices/screen-compat-mode.html)运行。声明应用程序支持的屏幕尺寸，应该在manifest文件中包括 [<supports-screens>](http://developer.android.com/guide/topics/manifest/supports-screens-element.html)元素。

* **为不同的屏幕尺寸提供不同的布局**

默认情况下，Android重新调整应用程序的布局，以适应当前的设备屏幕。在大多数情况下，这工作得很好。在其他情况下，用户界面可能看起来不太好，可能需要为不同屏幕尺寸的作调整。例如，在大屏幕上，可能要调整某些元素的位置和大小，充分利用额外的屏幕空间，或在一个较小的屏幕上，可能需要调整大小，让所有元素都可以在屏幕上显示。可以使用限定符来提供尺寸相关的资源，这些限定符包括small, normal, large, 和 xlarge。例如，一个超大大屏幕的布局，应该在layout-xlarge/。

从Android 3.2（API Level 13）开始，上述尺寸组已被弃用，应该使用sw<N>dp 限定符定义布局资源所需的最小可用宽度。例如，如果多窗格平板布局至少需要600dp屏幕的宽度，应该放置在layout-sw600dp/下。使用新技术来声明布局资源在 [为Android 3.2设计平板布局](#平板电脑设计布局)部分有进一步讨论。

* **为不同的屏幕尺寸提供不同的bitmap drawable**

默认情况下，Android缩放bitmap drawables（.png, .jpg,和.gif文件）和Nine-Patch drawables（.9.png 文件），使它们在每台设备上呈现合适的物理尺寸。

例如，如果应用程序只为基准，中型屏幕密度（MDPI）屏幕提供了bitmap drawable，那么系统将会在高密度屏幕上放大它们，而在低密度屏幕上缩小它们。这种缩放可能会让位图产生失真。

为了让位图最好的展示，应该为不同屏幕密度提供不同分辨率的位图。

可以使用限定符来提供密度相关的资源，这些限定符包括ldpi (low), mdpi (medium), hdpi (high), 和 xhdpi (extra high)。例如，为高密度屏幕提供的位图应该放在drawable-hdpi/下。

与广义的尺寸和密度相对应的尺寸和密度限定符在上文的[屏幕支持范围](#屏幕支持范围)中有描述。

**注:** 如果不熟悉配置限定符和系统如何使用它们来请求选择性资源，阅读[Providing Alternative Resources](http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources)获取更多信息。

在运行时，系统为任何给定资源通过以下步骤来尽可能确保它们在当前屏幕的显示：

1. 系统使用适当的可选资源

当前屏幕的大小和密度的基础上，系统使用应用程序提供的任意尺寸和密度相关资源。例如，设备有一个高密度的屏幕并且应用程序请求一个drawable 资源，系统会寻找一个最匹配设备配置的drawable 资源目录。与其它可选资源相比，以hdpi 为限定符的资源目录（比如drawable-hdpi/）会最佳匹配，所以系统使用该目录下的drawable资源。

1. 如果没有匹配的资源是可用的，系统将使用默认的资源并且对其进行缩放来适应当前的屏幕尺寸和密度。

“默认”的资源是那些没有配置限定符的资源。例如，在drawable/ 下的就是是默认的资源。系统假定默认资源是为基线屏幕尺寸和密度而设计的。正因为如此，系统对其采取缩放是比较可取的。

然而，当系统试图寻找一个密度相关的资源但是在密度相关的目录下没有找到时，系统并不总是使用默认资源。系统可能会改用其他密度相关的资源之一，以提供更好的缩放结果。 例如，当寻找一个低密度资源而没找到时，系统倾向于缩小高密度资源，因为系统可以通过缩放因子0.5轻松地将高密度资源缩小为低密度，这种缩放要比通过缩放因子0.75将中密度资源缩小为低密度不易失真。

更多信息请参阅[How Android Finds the Best-matching Resource](http://developer.android.com/guide/topics/resources/providing-resources.html" \l "BestMatch)。

**使用配置限定符**

Android支持多种配置限定符，来控制系统如何根据当前设备屏幕的特点选择可选的资源。配置限定符是一个字符串，可以在Andr​​oid项目追加到资源目录，并指定配置资源的内部设计。

使用配置的限定符：

1. 在工程的res/目录下创建一个新的目录，并将其以如下格式命名：<resources\_name>-<qualifier>
   * <resources\_name> 是标准的资源名 (比如drawable 或者 layout).
   * <qualifier> 是一个配置限定符， 如下面的表1所示， 它限定了这些资源可以用于的屏幕配置(比如 hdpi 或者 xlarge).

可以一次使用多个 <qualifier> —使用破折号将多个限定符分开。

1. 在这个新目录保存适当的配置相关资源。资源文件命名务必与默认的资源文件命名一致。

比如, xlarge 是一个超大屏幕的限定符。当追加此字符串到一个资源目录名（比如layout-xlarge），这就告诉系统这些资源是要用在拥有超大屏幕的设备上的。

**表 1.** 为不同屏幕配置提供具体资源的配置限定符

|  |  |  |
| --- | --- | --- |
| **Screen characteristic** | **Qualifier** | **Description** |
| Size | small | Resources for *small* size screens. |
| normal | Resources for *normal* size screens. (This is the baseline size.) |
| large | Resources for *large* size screens. |
| xlarge | Resources for *extra large* size screens. |
| Density | ldpi | Resources for low-density (*ldpi*) screens (~120dpi). |
| mdpi | Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.) |
| hdpi | Resources for high-density (*hdpi*) screens (~240dpi). |
| xhdpi | Resources for extra high-density (*xhdpi*) screens (~320dpi). |
| nodpi | Resources for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density. |
| tvdpi | Resources for screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. If you find it necessary to provide tvdpi resources, you should size them at a factor of 1.33\*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi. |
| Orientation | land | Resources for screens in the landscape orientation (wide aspect ratio). |
| port | Resources for screens in the portrait orientation (tall aspect ratio). |
| Aspect ratio | long | Resources for screens that have a significantly taller or wider aspect ratio (when in portrait or landscape orientation, respectively) than the baseline screen configuration. |
| notlong | Resources for use screens that have an aspect ratio that is similar to the baseline screen configuration. |

**注：**对于Android 3.2以及以上版本，请参阅 [为Android 3.2设计平板布局](#平板电脑设计布局)获取关于新的配置限定符的更多信息（不要使用表1中的尺寸限定符）

欲了解更多有关如何将这些限定符与实际的屏幕尺寸和密度相对应的信息，请看本文档前面的[屏幕支持范围](#屏幕支持范围)

例如，下面是一个应用程序的资源列表，它为不同的屏幕尺寸提供不同的布局，并且为中，高，超高密度的屏幕提供不同的bitmap drawable。

res/layout/my\_layout.xml    // layout for normal screen size ("default")  
res/layout-small/my\_layout.xml       // layout for small screen size  
res/layout-large/my\_layout.xml       // layout for large screen size  
res/layout-xlarge/my\_layout.xml  // layout for extra large screen size  
res/layout-xlarge-land/my\_layout.xml // layout for extra large in landscape orientation  
  
res/drawable-mdpi/my\_icon.png        // bitmap for medium density  
res/drawable-hdpi/my\_icon.png        // bitmap for high density  
res/drawable-xhdpi/my\_icon.png       // bitmap for extra high density

欲了解更多有关如何使用选择性资源和配置限定符的完整列表（不只是屏幕配置）的信息，请参阅[Providing Alternative Resources](http://developer.android.com/guide/topics/resources/providing-resources.html" \l "AlternativeResources)。

注意，Android系统在运行时选择使用哪些资源时，它使用某些逻辑来决定“最佳匹配”的资源。也就是说，使用的限定符没有必要完全匹配当前的屏幕配置。具体来说，当基于尺寸限定符来选择资源时，如果没有更好的匹配，系统会选择使用为比当前屏幕小的屏幕设计的资源（例如，一个大屏如果需要，可能会使用正常屏的资源）。但是，如果只有比当前屏幕尺寸大的资源，那么系统将不会使用这些大的资源，应用程序将会因无从选择而崩溃。（例如，如果所有的资源都是xlarge 限定符的，但是设备是一个normal-size的设备）。

更多关于系统如何选择资源的信息，请参阅 [How Android Finds the Best-matching Resource](http://developer.android.com/guide/topics/resources/providing-resources.html#BestMatch)。

**提示**：如果有一些绘制资源，不想让系统进行缩放（也许是因为想要在程序运行时进行手动调整），应该将其放置nodpi 配置限定符的目录下。这个限定符的资源被视为密度无关，系统将不能扩展。

**设计选择性的布局和图片**

替代资源的类型，取决于应用程序的需求。通常，应该使用尺寸和方向限定符提供选择性的布局资源，使用密度限定符提供选择性的图片资源。以下部分分别概括了该如何使用尺寸和密度限定符提供选择性的布局和图片。

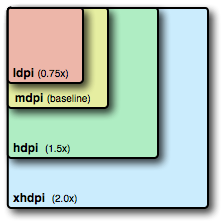
**选择性的布局**

应该确保应用程序的布局：

* 适合在小屏幕上（确保可以实际使用）
* 在大屏幕上，确保充分利用额外的屏幕空间
* 适应竖屏和横屏

如果用户界面，需要让位图来适应视图的大小（如一个按钮的背景图片），应该使用[Nine-Patch](http://developer.android.com/guide/topics/graphics/2d-graphics.html#nine-patch) 位图文件。Nine-Patch文件基本上是一个PNG文件它可以在特定的二维地区伸展。当系统需要拉伸位图所在的视图时，系统会拉伸Nine-Patch文件，但是仅仅拉伸的是指定区域。正因为如此，不需要提供不同的屏幕尺寸不同的图片，因为Nine-Patch的位图，可以调整到任何尺寸。然而，应该为不同的屏幕密提供不同的Nine-Patch文件。

**选择性的图片**



**Figure 4.** 支持各密度的位图的相对大小

几乎每个应用程序都应当为不同的屏幕密度提供图片资源，因为几乎每一个应用程序都拥有一个启动图标，而改图标应当在不同密度的屏幕上显示良好。

**注:** 只需提供密度相关的位图文件(.png, .jpg, 或者 .gif) 和 Nine-Path 文件(.9.png)。如果需要使用XML来定义形状，颜色或者可绘制的资源，应当在默认的drawable 目录备份一份。

要创建不同密度的选择性的图片，应该在四个广义密度之间遵循3:4:6:8的缩放比例。比如，一个中密度下为48x48像素的位图（应用程序启动图标），与之对应的所有不同的尺寸应当是：

* 低密度为36x36
* 中密度为48x48
* 高密度为72x72
* 超高密度为96x96

更多关于设计图标的信息，请参阅 [Icon Design Guidelines](http://developer.android.com/guide/practices/ui_guidelines/icon_design.html)，它涵盖了各种位图的尺寸信息，比如启动图标，菜单图标，状态栏图标，tab 图标等等。

**为Android 3.2设计平板布局**

对于第一代运行Android 3.0的平板电脑来说，声明平板布局的适合方式是使用xlarge 限定符（比如res/layout-xlarge/）。为了适应其他类型的平板电脑和屏幕尺寸，Android 3.2引入了一种新的为分散屏幕尺寸定义资源的方式。这种新技术是基于布局所需的空间数量（比如600dp宽），而不是试图让布局去符合广义的尺寸分类（比如大或者超大）。

**使用新的尺寸限定符**

见表2，这些新的限定符，比传统的屏幕尺寸分类（小，中，大，和超大）能提供更多的对于定义应用程序支持的屏幕的控制。

**注:** 使用这​​些限定符指定的大小不是实际的屏幕尺寸。相反，是可以用于活动窗口的宽度或者高度尺寸，是以dp为单位的。系统会为系统UI使用一些屏幕空间（比如系统栏或者状态栏），所以一些屏幕空间对应用程序是不可用的。因此，声明的尺寸应当明确的是activity所需的尺寸—当系统声明应当提供多少空间给布局时，它会计算任何被系统UI占据的空间。 同时注意， [Action Bar](http://developer.android.com/guide/topics/ui/actionbar.html)被认为是应用程序的一部分空间，尽管布局没有声明它，因此它减少了可用的布局空间，你必须考虑在设计之内。

**Table 2.** 屏幕尺寸的新的配置限定符（在Android 3.2中引入）

|  |  |  |
| --- | --- | --- |
| **Screen configuration** | **Qualifier values** | **Description** |
| smallestWidth | sw<N>dp  Examples: sw600dp sw720dp | The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (you may also think of it as the "smallest possible width" for the screen). You can use this qualifier to ensure that, regardless of the screen's current orientation, your application's has at least <N> dps of width available for it UI.  For example, if your layout requires that its smallest dimension of screen area be at least 600 dp at all times, then you can use this qualifer to create the layout resources, res/layout-sw600dp/. The system will use these resources only when the smallest dimension of available screen is at least 600dp, regardless of whether the 600dp side is the user-perceived height or width. The smallestWidth is a fixed screen size characteristic of the device; **the device's smallestWidth does not change when the screen's orientation changes**.  The smallestWidth of a device takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the screen that account for space along the axis of the smallestWidth, the system declares the smallestWidth to be smaller than the actual screen size, because those are screen pixels not available for your UI.  This is an alternative to the generalized screen size qualifiers (small, normal, large, xlarge) that allows you to define a discrete number for the effective size available for your UI. Using smallestWidth to determine the general screen size is useful because width is often the driving factor in designing a layout. A UI will often scroll vertically, but have fairly hard constraints on the minimum space it needs horizontally. The available width is also the key factor in determining whether to use a one-pane layout for handsets or multi-pane layout for tablets. Thus, you likely care most about what the smallest possible width will be on each device. |
| Available screen width | w<N>dp  Examples: w720dp w1024dp | Specifies a minimum available width in dp units at which the resources should be used—defined by the <N> value. The system's corresponding value for the width changes when the screen's orientation switches between landscape and portrait to reflect the current actual width that's available for your UI.  This is often useful to determine whether to use a multi-pane layout, because even on a tablet device, you often won't want the same multi-pane layout for portrait orientation as you do for landscape. Thus, you can use this to specify the minimum width required for the layout, instead of using both the screen size and orientation qualifiers together. |
| Available screen height | h<N>dp  Examples: h720dp h1024dp etc. | Specifies a minimum screen height in dp units at which the resources should be used—defined by the <N> value. The system's corresponding value for the height changes when the screen's orientation switches between landscape and portrait to reflect the current actual height that's available for your UI.  Using this to define the height required by your layout is useful in the same way as w<N>dp is for defining the required width, instead of using both the screen size and orientation qualifiers. However, most apps won't need this qualifier, considering that UIs often scroll vertically and are thus more flexible with how much height is available, whereas the width is more rigid. |

虽然使用这些限定符可能看起来比使用屏幕尺寸分组似乎更复杂，一旦确定用户界面的要求，它实际上应该是更简单。当设计UI时，关心的主要事情是当应用程序在手机式UI和平板式UI之间迁移时它的实际尺寸。

更多关于尺寸配置限定符的讨论，请参阅[Providing Resources](http://developer.android.com/guide/topics/resources/providing-resources.html" \l "SmallestScreenWidthQualifier)

**配置实例**

为了您针对不同类型的设备进行设计，这里是典型的屏幕宽度的一些数字：

* 320dp: 一个典型的手机屏幕 (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, 等).
* 480dp: 像Streak一样的中立平板屏幕 (480x800 mdpi).
* 600dp: 7”平板 (600x1024 mdpi).
* 720dp: 10”平板 (720x1280 mdpi, 800x1280 mdpi, 等).

使用表2中的尺寸限定符，应用程​​序可以使用任何您想要的宽度和/或高度为手机和平板之间切换不同的布局资源。比如，如果600dp是平板布局所支持的最小可用宽度，可以提供这两套布局：

res/layout/main\_activity.xml           # For handsets  
res/layout-sw600dp/main\_activity.xml   # For tablets

在这种情况下，为了让平板布局实用，可用屏幕空间的最小宽度必须为600dp。

要进一步定制UI区分大小，如7“和10”平板的情况下，可以定义额外的最小宽度布局：

res/layout/main\_activity.xml           # For handsets (smaller than 600dp available width)  
res/layout-sw600dp/main\_activity.xml   # For 7” tablets (600dp wide and bigger)  
res/layout-sw720dp/main\_activity.xml   # For 10” tablets (720dp wide and bigger)

注意，前面的两个例子资源使用“最小宽度”限定符，sw<N>dp，它指定屏幕两边的最小值，无论设备的当前方向。因此，使用sw<N>dp是简单的通过忽略屏幕方向的方式来指定布局的整体屏幕尺寸。

然而，在某些情况下，可能重要的是布局究竟有多大的宽度或高度目前是可用的。例如，有两个并排的布局，每当屏幕宽度至少提供600dp时使用，不论设备是否是在横向或纵向。在这种情况下，资源看起来像这样：

res/layout/main\_activity.xml         # For handsets (smaller than 600dp available width)  
res/layout-w600dp/main\_activity.xml  # Multi-pane (any screen with 600dp available width or more)

请注意，第二组使用“可用宽度”限定符，w<N>dp。这样，一台设备可能根据屏幕的方向会使用这两种布局（如果可用宽度在一个方向至少600dp而在另一个方向小于600dp）。

如果关注的是可用高度，那么同样可以使用h<N>dp限定符。甚至可以结合使用w<N>dp 和 h<N>dp 限定符。

**声明屏幕尺寸支持**

一旦已经为不同的屏幕尺寸实现了布局，同样重要的是在应用程序的manifest文件中声明应用程序支持哪些屏幕。

随着屏幕尺寸的新的配置限定符，Android 3.2引入了[<supports-screens>](http://developer.android.com/guide/topics/manifest/supports-screens-element.html) 元素的新属性。

[android:requiresSmallestWidthDp](http://developer.android.com/guide/topics/manifest/supports-screens-element.html#requiresSmallest)

指定的最低宽度。 smallestWidth是指必须提供给应用程序的UI最短尺寸的屏幕空间（单位dp），也就是说，最短的可用屏幕的两个维度。因此，为了让一个设备与应用程序兼容，设备的smallestWidth必须等于或大于这个值。 （通常情况下，该值是布局支持测“最小宽度”，无论当前屏幕的方向。）

例如，如果应用程序仅用于最小可用宽度是600dp的平板式设备：

<manifest ... >  
    <supports-screens android:requiresSmallestWidthDp="600" />  
    ...  
</manifest>

不过，如果应用程序支持Andr​​oid支持的所有屏幕尺寸（如426dp x 320dp这么小），那么并不需要声明这个属性，因为应用程序需要的最小宽度在任何设备上都是最小的。

注意：Android系统并不关注这个属性，所以它不会影响应用程序在运行时的行为。相反，它被用于过滤应用程序，如Google Play（谷歌播放）。然而，目前谷歌播放不支持过滤（基于Android 3.2）这个属性，所以如果应用程序不支持小屏幕，应该继续使用其他的大小属性。

[android:compatibleWidthLimitDp](http://developer.android.com/guide/topics/manifest/supports-screens-element.html#compatibleWidth)

这个属性允许通过指定应用程序支持的最大"smallest width"来让[screen compatibility mode](http://developer.android.com/guide/practices/screen-compat-mode.html) （屏幕兼容模式）成为用户可选的特性。如果设备的可用屏幕最小的一边大于该值，用户仍然可以安装该应用程序，但提供在屏幕兼容性模式运行。默认情况下，屏幕兼容模式是无效的，而且布局会去调整适应普通的屏幕，但是系统栏会提供一个按钮来允许用户切换屏幕兼容模式开关。

**注**：如果应用程序的布局适当调整以适应大屏幕，则不需要使用这个属性。建议避免使用这个属性，而应当按照本文的建议确保布局调整以适应大屏幕。

[android:largestWidthLimitDp](http://developer.android.com/guide/topics/manifest/supports-screens-element.html#largestWidth)

这个属性允许通过指定应用程序支持的最大的"smallest width"来强制启用 [screen compatibility mode](http://developer.android.com/guide/practices/screen-compat-mode.html)屏幕兼容模式。如果设备的可用屏幕最小的一边大于该值，应用程序会强制运行在屏幕兼容模式，用户无法干预。

**注**：如果应用程序的布局适当调整以适应大屏幕，则不需要使用这个属性。建议避免使用这个属性，而应当按照本文的建议确保布局调整以适应大屏幕。

**注意**：为Android 3.2及以上版本开发时，不应该使用上面列出的属性结合旧的屏幕大小属性。同时使用新属性和旧属性，可能会导致意想不到的行为。

**最佳实践**

支持多屏幕的目标是创建一个应用程序可以正常并且很好地运行在Android支持的任何的广义屏配置。前几节提供有关Android是如何适应应用程序的屏幕配置，以及如何自定义应用程序在不同的屏幕配置上的展现。本节提供了一些额外的技巧和技术的概述，以帮助确保应用程序适配不同的屏幕配置。

这里是一个有关如何可以确保应用程序在不同的屏幕上正确显示的快速清单：

1. 当在XML布局文件中定义尺码时，使用 wrap\_content, fill\_parent, 或者 dp 单位
2. 不要在应用程序代码中使用硬编码的像素值
3. 不要使用 AbsoluteLayout (它已被废弃)
4. 为不同的屏幕密度提供可选的位图

以下部分提供更多细节。

**1.当在XML布局文件中定义尺码时，使用 wrap\_content, fill\_parent, 或者 dp 单位**

当为XML布局文件定义[android:layout\_width](http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html" \l "attr_android:layout_width) 和 [android:layout\_height](http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html" \l "attr_android:layout_height)时，使用"wrap\_content", "fill\_parent" 或者 dp单位，可以确保视图在当前屏幕上有一个合适的尺寸。

例如，一个layout\_width="100dp"的视图在中密度屏上占用100像素，而在高密度屏上则是150像素，所以视图在屏幕上占用大约相近的物理空间。

同样，应该更倾向于使用sp （scale-independent pixel）来定义文本大小。 sp 缩放因子和dp.一样取决于用户设置和系统缩放的大小。

**2.不要在应用程序代码中使用硬编码的像素值**

不作过多解释。

如果应用程序操纵位图或在​​运行时的像素值，请参阅[Additional Density Considerations](http://developer.android.com/guide/practices/screens_support.html#DensityConsiderations)。

**3. 不要使用 AbsoluteLayout**

不同于其他布局部件，[AbsoluteLayout](http://developer.android.com/reference/android/widget/AbsoluteLayout.html)强制使用固定的位置去摆放子视图，这很容易导致在不同的显示器的用户界面不能好好工作。 正因为如此，[AbsoluteLayout](http://developer.android.com/reference/android/widget/AbsoluteLayout.html)在Android 1.5（API等级3）被废弃。

应该使用[RelativeLayout](http://developer.android.com/reference/android/widget/RelativeLayout.html)，它使用相对定位来摆放子视图。例如，可以指定一个按钮部件让它显示在文本部件的右侧。

**4. 使用基于尺寸和密度的资源**

如果想严格控制应用程序在不同屏幕配置上的展示，那就在配置相关的资源目录下调整布局和图片。例如，考虑到想要在中等和高密度显示屏上用到的图标，只需创建两个大小不同（例如中密度的为100x100，高密度的为150X150）的图标，并使用适当的限定放置在相应目录下：

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

注：如果目录名中没有定义密度限定符，系统假定该目录中的资源为基准密度而设计，并会为其他密度作适当扩展。

更多关于配置限定符的信息，请参阅[使用配置限定符](#使用配置限定符)

**关于密度的额外思考**

本节将介绍更多的Andr​​oid在不同的屏幕密度上如何处理位图的缩放，以及如何可以进一步控制位图在不同屏幕密度上的绘制。本节信息对于大多数的应用程序是不重要的，除非在不同屏幕密度上运行应用程序时遇到问题或者应用程序需要操作图像。

为了更好地了解如何在运行时操作图像时支持多密度，应该了解，系统有助于确保位图的适当缩放，表现在以下方面：

1. *预缩放资源 (比如 bitmap drawables)*

基于当前屏幕的密度，系统不加缩放地使用应用程序的任何尺寸或密度相关资源。如果资源不是以正确的密度提供的，系统会加载默认的资源并且按需要根据当前屏幕密度进行缩放。系统假定默认资源（那些在没有配置限定符的目录下的资源）是为基准屏幕密度（mdpi）设计的，除非他们是从一个特定密度的资源目录加载。因此，预缩放就是系统为当前屏幕密度进行适当的缩放资源。

如果请求预缩放资源的尺寸，系统将返回缩放后的尺寸。例如位图，为mdpi屏幕设计的50 x50像素的位图在hdpi的屏幕上被放大到75x75像素，那么系统报告的就是75x75像素。

有一些情况下，可能希望Android不进行预缩放。避免预缩放的最简单的方法，就是把资源放在以nodpi 配置限定符标识的目录下，比如：

res/drawable-nodpi/icon.png

这样，当系统选用该文件夹的icon.png位图时，不会基于当前屏幕密度进行扩展。

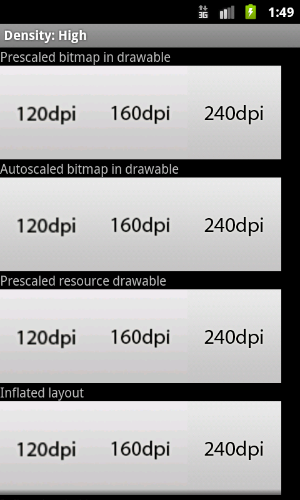
1. *像素尺寸和坐标的自动缩放*

应用程序可以通过在manifest中设置[android:anyDensity](http://developer.android.com/guide/topics/manifest/supports-screens-element.html#any) 为“false”或者在程序中设置[Bitmap](http://developer.android.com/reference/android/graphics/Bitmap.html)的[inScaled](http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html" \l "inScaled)为"false"这种来禁用预缩放。在这种情况下，系统在绘制时自动调整任何绝对的像素坐标和像素尺寸值。它这样做，以确保像素定义屏幕元素仍然与在基准屏幕密度（mdpi）下显示相近的物理尺寸。系统为应用程序透明的处理这个比例并且报告缩放过的像素给应用程序，而不是物理像素尺寸。

通常情况下，不应该禁用预缩放。支持多个屏幕的最好办法是上文提到的[如何支持多屏幕](#如何支持多屏幕)。

如果应用程序操纵位图，或以一些其他的方式与屏幕上的像素进行直接交互，可能需要采取额外的步骤，以支持不同的屏幕密度。例如，如果通过计算手机经过的像素数来响应触摸手势，则需要使用适当的密度无关的像素值，而不是实际像素。

**在运行时创建的缩放位图对象**



**Figure 5.** 预缩放和自动缩放位图的比较, 来自 [ApiDemos](http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/DensityActivity.html).

如果应用程序创建一个内存位图（[Bitmap](http://developer.android.com/reference/android/graphics/Bitmap.html) 对象），系统会假设该位图是专为基准密度屏幕设计的并且在绘制时进行自动缩放。系统当位图有不明密度属性时，将 “自动缩放”运用于一个Bitmap位图。如果没有正确地统计当前设备的屏幕密度，并指定位图的密度属性，自动缩放会导致缩放失真，就像没有提供可选性资源一样。

为了控制在运行时创建的位图是否被缩放，可以用[setDensity()](http://developer.android.com/reference/android/graphics/Bitmap.html#setDensity(int))来指定位图的密度，通过传递一个来自[DisplayMetrics](http://developer.android.com/reference/android/util/DisplayMetrics.html)的密度常量，比如[DENSITY\_HIGH](http://developer.android.com/reference/android/util/DisplayMetrics.html#DENSITY_HIGH) 或者 [DENSITY\_LOW](http://developer.android.com/reference/android/util/DisplayMetrics.html#DENSITY_LOW)

如果使用[BitmapFactory](http://developer.android.com/reference/android/graphics/BitmapFactory.html)创建一个[Bitmap](http://developer.android.com/reference/android/graphics/Bitmap.html) ，如从一个文件或流，可以使用[BitmapFactory.Options](http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html)来定义位图属性，该属性决定了系统是否或者如何缩放位图。

图5演示了预缩放和自动缩放在高密度屏上加载低（120），中（160）和高（240）密度位图的结果。差异是微妙的，因为所有位图被缩放以匹配当前屏幕密度，但是缩放的位图呈现细微的区别，这取决于它们在绘制时时预缩放的还是自动缩放的。可以在[ApiDemos](http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/DensityActivity.html)代码中找到这个示例应用程序，它演示了如何使用预缩放和自动缩放位图。

**注:** 在Android 3.0及以上版本，由于在图形架构的改进，预缩放和自动缩放位图之间没有可感知的差异，

**dp单位转换为像素单位**

在某些情况下，需要用dp 来表示尺寸，然后将它们转换为像素。想象一个应用程序，当用户手指移动至少16像素，则形成一个滚动或者抛掷手势。在基准屏幕上，用户必须在手势形成之前移动16个像素/ 160 dpi，等于一英寸的十分之一（2.5毫米）。在高密度显示器（240dpi）的设备上，用户必须移动16个像素/ 240 dpi，这等于一英寸的十五分之一（1.7毫米）。距离要短得多，因此，应用程序似乎对用户更加敏感。

为了解决这个问题，在代码中手势门槛必须用dp 来表示，然后转化为实际的像素。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()](http://developer.android.com/reference/android/content/ContextWrapper.html#getResources()).[getDisplayMetrics()](http://developer.android.com/reference/android/content/res/Resources.html#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](http://developer.android.com/reference/android/util/DisplayMetrics.html" \l "density)域指定缩放因子，使用该缩放因子根据当前的屏幕密度来把dp 转换为像素。在一个中等密度的屏幕上，[DisplayMetrics.density](http://developer.android.com/reference/android/util/DisplayMetrics.html#density)等于1.0；在一个高密度的屏幕上，它等于1.5；在一个超高密度屏幕上，它等于2.0；在低密度屏幕上，它等于0.75。用dp乘以这个数字就得到当前屏幕的实际像素数值。 （当转换为整数时，添加0.5f 四舍五入到最接近的整数）。有关详细信息，请参阅的[DisplayMetrics](http://developer.android.com/reference/android/util/DisplayMetrics.html)类。

然而，对这类事件，应该使用 [ViewConfiguration](http://developer.android.com/reference/android/view/ViewConfiguration.html)预缩放的配置值，而不是定义一个任意的门槛。

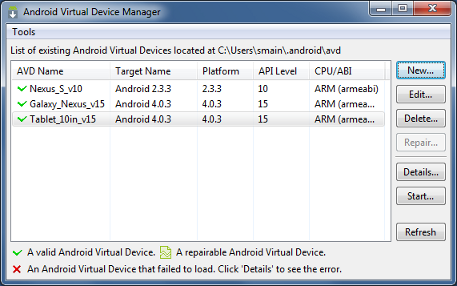
**使用预缩放的配置值**

可以使用[ViewConfiguration](http://developer.android.com/reference/android/view/ViewConfiguration.html) 类访问Android系统的一般的的距离，速度和时间。例如，可以通过[getScaledTouchSlop()](http://developer.android.com/reference/android/view/ViewConfiguration.html#getScaledTouchSlop())获得框架使用的滚动门槛；

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

[ViewConfiguration](http://developer.android.com/reference/android/view/ViewConfiguration.html) 类中以getScaled 为前缀的方法可以保证返回可以正确显示的像素值，不论当前的屏幕密度。

**如何在多屏幕上测试应用程序**



**Figure 6.** 用来测试屏幕支持的一组AVD。

发布应用程序之前，应该在所有支持的屏幕尺寸和密度上进行彻底的测试。 Android SDK中包含可以使用的模拟器皮肤，它复制常见的屏幕配置的大小和密度，应用程序可能运行于这些皮肤。也可以以任何特定的屏幕特点修改默认的大小，密度，和模拟器皮肤。使用模拟器的皮肤和额外的自定义配置，可以测试任何可能的屏幕配置，而不用买各种设备。

为了搭建一个测试应用程序屏幕支持的环境，应该创建一系列AVD（Android虚拟设备），使用模拟器的外观和屏幕配置，模拟应用程序支持的屏幕尺寸和密度。这样做，可以使用AVD管理器创建AVDS，然后使用图形界面启动它们。

启动Andr​​oid SDK管理器，从Andr​​oid SDK目录执行SDK Manager.exe 的（仅适用于Windows）或从<sdk>/tools/ 目录执行android （所有平台）。图6显示了AVD管理器中的一组AVD。

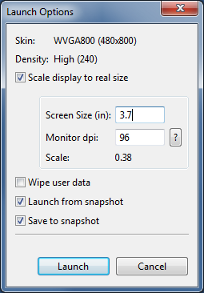
表3显示了Android SDK中的各种仿真皮肤，可以用它来模拟一些最常见的屏幕配置。

欲了解更多有关创建和使用的AVD来测试应用的详细信息，请参阅[Managing AVDs with AVD Manager](http://developer.android.com/guide/developing/devices/managing-avds.html)。

**Table 3.** Android SDK中模拟器皮肤提供的各种屏幕配置和典型的分辨率。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Low density (120), *ldpi*** | **Medium density (160), *mdpi*** | **High density (240), *hdpi*** | **Extra high density (320), *xhdpi*** |
| ***Small* screen** | **QVGA (240x320)** |  | 480x640 |  |
| ***Normal* screen** | **WQVGA400 (240x400)**  **WQVGA432 (240x432)** | **HVGA (320x480)** | **WVGA800 (480x800)**  **WVGA854 (480x854)**  600x1024 | 640x960 |
| ***Large* screen** | **WVGA800\*\* (480x800)**  **WVGA854\*\* (480x854)** | **WVGA800\* (480x800)**  **WVGA854\* (480x854)**  600x1024 |  |  |
| ***Extra Large* screen** | 1024x600 | **WXGA (1280x800)**† 1024x768 1280x768 | 1536x1152 1920x1152  1920x1200 | 2048x1536 2560x1536  2560x1600 |
|  | | | | |

查看支持任何屏幕配置的活动设备的相关数值，请参阅[Screen Sizes and Densities](http://developer.android.com/resources/dashboard/screens.html)



**Figure 7.** 当从AVD管理器启动AVD时，可以设置的尺寸和密度选项

还建议在这样的模拟器上去测试应用程序，此模拟器被配置为物理尺寸和实际设备相近。这样做，需要电脑显示器的大概的DPI密度，（例如，30“戴尔显示器具有大约96 dpi的密度）。当从AVD管理器启动一个AVD时，可以在Launch Options中为模拟器和监视器指定dpi，如图7所示。

如果想要在一个不被内置皮肤所支持的分辨率或者密度的屏幕上测试应用程序，可以用自定义的分辨率和密度创建一个AVD。当从AVD管理器创建一个AVD时，定义分辨率，而不是选择内置皮肤。

如果从命令行启动AVD，可以使用-scale例如选项来指定模拟器规模。例如：

emulator -avd <avd\_name> -scale 96dpi

为了细化模拟器的尺寸，可以替代-scale选项为一个0.1到3之间的数字，它代表期望的缩放因子。

**注**：本文译自android开发者网站，原文请参考<http://developer.android.com/guide/practices/screens_support.html>