支持多种屏幕

内容快览

- Android 可在具有不同屏幕尺寸和密度的设备上运行。
- 显示应用的屏幕可影响其用户界面。
- 系统会处理大多数工作，使您的应用适应当前屏幕。
- 您应该创建屏幕特定的资源，以精确控制 UI。

本文内容

相关示例

- 多种分辨率

另请参阅

- 视同 Web Designer
- 提供备用资源
- 图标设计指南
- 管理虚拟设备

Android 可在各种具有不同屏幕尺寸和密度的设备上运行。对于应用，Android 系统在不同设备中提供一致的开发环境，可以处理大多数工作，将每个应用的用户界面调整为适应其显示的屏幕。 同时，系统提供 API，可用于控制应用适用于特定屏幕尺寸和密度的 UI，以针对不同屏幕配置优化 UI 设计。 例如，您可能想要不同于手机 UI 的平板电脑 UI。

虽然系统为使您的应用适用于不同的屏幕，会进行缩放和大小调整，但您应针对不同的屏幕尺寸和密度优化应用。 这样可以最大程度优化所有设备上的用户体验，用户会 认为您的应用实际上是专为*他们的*设备而设计，而不是 简单地拉伸以适应其设备屏幕。

按照本文档所述的做法，您可以创建 正常显示的应用，然后使用 一个 `.apk` 文件在所有支持的屏幕配置中提供优化的用户体验。

> **注**：本文档中的信息假设您的 应用设计用于 Android 1.6（API 级别 4）或更高级别。如果您的应用只支持 Android 1.5 或更低版本，请先阅读适用于 Android 1.5 的策略。
>
> 另请注意，**Android 3.2 引入了新的 API**，可用于更 精确地控制应用用于不同屏幕尺寸的布局资源。如果您要开发针对平板电脑优化的应

> 用，这些新 功能特别重要。 有关详情，请参阅声明 Android 3.2 的平板电脑布局相关章节。

屏幕支持概览

本节概述 Android 对多种屏幕的支持，包括： 本文档和 API 中所用术语和概述的简介、 系统支持的屏幕配置摘要，以及 API 和基本 屏幕兼容性功能的概述。

术语和概念

屏幕尺寸

> 按屏幕对角测量的实际物理尺寸。
>
> 为简便起见，Android 将所有实际屏幕尺寸分组为四种通用尺寸：小、 正常、大和超大。

屏幕密度

> 屏幕物理区域中的像素量；通常称为 dpi（每英寸 点数）。例如， 与“正常”或“高”密度屏幕相比，“低”密度屏幕在给定物理区域的像素较少。
>
> 为简便起见，Android 将所有屏幕密度分组为六种通用密度： 低、中、高、超高、超超高和超超超高。

方向

> 从用户视角看屏幕的方向，即横屏还是 竖屏，分别表示屏幕的纵横比是宽还是高。请注意， 不仅不同的设备默认以不同的方向操作，而且 方向在运行时可随着用户旋转设备而改变。

分辨率

> 屏幕上物理像素的总数。添加对多种屏幕的支持时， 应用不会直接使用分辨率；而只应关注通用尺寸和密度组指定的屏幕 尺寸及密度。

密度无关像素 (dp)

> 在定义 UI 布局时应使用的虚拟像素单位，用于以密度无关方式表示布局维度 或位置。
>
> 密度无关像素等于 160 dpi 屏幕上的一个物理像素，这是 系统为“中”密度屏幕假设的基线密度。在运行时，系统 根据使用中屏幕的实际密度按需要以透明方式处理 dp 单位的任何缩放 。dp 单位转换为屏幕像素很简单： `px = dp * (dpi / 160)`。 例如，在 240 dpi 屏幕上，1 dp 等于 1.5 物理像素。在定义应用的 UI 时应始终使用 dp 单位 ，以确保在不同密度的屏幕上正常显示 UI。

支持的屏幕范围

从 Android 1.6（API 级别 4）开始，Android 支持多种屏幕尺寸和密度，反映设备可能具有的多种不同屏幕配置。 您可以使用 Android 系统的功能优化应用在各种屏幕配置下的用户界面 ，确保应用不仅正常渲染，而且在每个屏幕上提供 最佳的用户体验。

为简化您为多种屏幕设计用户界面的方式，Android 将实际屏幕尺寸和密度的范围 分为：

- 四种通用**尺寸**：*小*、*正常*、 *大* 和*超大*

> **注**：从 Android 3.2（API 级别 13）开始，这些尺寸组 已弃用，而采用根据可用屏幕宽度管理屏幕尺寸的 新技术。如果为 Android 3.2 和更高版本开发，请参阅声明适用于 Android 3.2 的平板电脑布局以了解更多信息。

- 六种通用的**密度**：
  - *ldpi*（低）~120dpi
  - *mdpi*（中）~160dpi
  - *hdpi*（高）~240dpi
  - *xhdpi*（超高）~320dpi
  - *xxhdpi*（超超高）~480dpi

- *xxxhdpi*（超超超高）~640dpi

通用的尺寸和密度按照基线配置（即*正常*尺寸和 *mdpi*（中）密度）排列。 此基线基于第一代 Android 设备 (T-Mobile G1) 的屏幕配置，该设备采用 HVGA 屏幕（在 Android 1.6 之前，这是 Android 支持的唯一屏幕配置）。

每种通用的尺寸和密度都涵盖一个实际屏幕尺寸和密度范围。例如， 两部都报告*正常*屏幕尺寸的设备在手动测量时，实际屏幕尺寸和 高宽比可能略有不同。类似地，对于两台报告 *hdpi* 屏幕密度的设备，其实际像素密度可能略有不同。 Android 将这些差异抽象概括到应用，使您可以提供为通用尺寸和密度设计的 UI，让系统按需要处理任何最终调整。 图 1 说明不同的尺寸和密度如何粗略归类为不同的尺寸 和密度组。

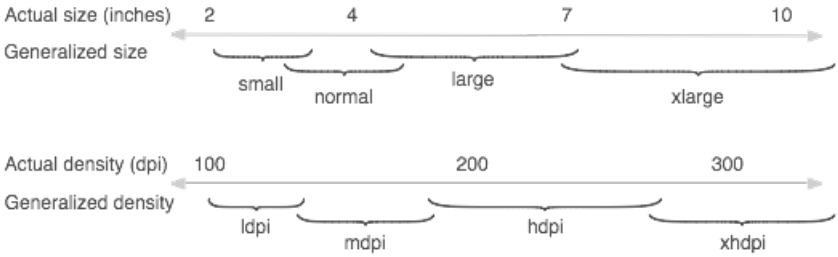

图 1. 说明 Android 如何将实际尺寸和密度粗略地 对应到通用的尺寸和密度（数据并不精确）。

在为不同的屏幕尺寸设计 UI 时，您会发现每种设计都需要 最小空间。因此，上述每种通用的屏幕尺寸都关联了系统定义的最低 分辨率。这些最小尺寸以“dp”单位表示 — 在定义布局时应使用相同的单位 — 这样系统无需担心屏幕密度的变化。

- *超大*屏幕至少为 960dp x 720dp
- *大*屏幕至少为 640dp x 480dp
- *正常*屏幕至少为 470dp x 320dp
- *小*屏幕至少为 426dp x 320dp

> **注**：这些最小屏幕尺寸在 Android 3.0 之前未正确定义，因此某些设备在正常屏幕与大屏幕之间变换时可能会出现分类错误的情况。 这些尺寸还基于屏幕的物理分辨率，因此设备之间可能不同 — 例如，具有系统状态栏的 1024x720 平板电脑因系统状态栏要占用空间，所以可供应用使用的空间要小一点。

要针对不同的屏幕尺寸和密度优化应用的 UI，可为任何通用的尺寸和密度提供备用资源。 通常，应为某些不同的屏幕尺寸提供替代布局，为不同的屏幕密度提供替代位图图像。 在运行时，系统会根据当前设备屏幕的通用 尺寸或密度对应用使用适当的资源。

无需为屏幕尺寸和 密度的每个组合提供备用资源。系统提供强大的兼容性功能，可处理在任何设备屏幕上 渲染应用的大多数工作，前提是您已经使用 可以适当调整大小的技术实现 UI（如下面的最佳做法所述）。

> **注**：定义设备通用屏幕 尺寸和密度的特性相互独立。例如，WVGA 高密度屏幕 被视为正常尺寸屏幕，因为其物理尺寸与 T-Mobile G1（Android 的第一代设备和基线屏幕配置）大约相同。另一方面，WVGA 中密度 屏幕被视为大尺寸屏幕。虽然它提供相同的分辨率（相同的 像素数），但 WVGA 中密度屏幕的屏幕密度更低，意味着每个像素 实际上更大，因此整个屏幕大于基线（正常尺寸）屏幕。

密度独立性

应用显示在密度不同的屏幕上时，如果它保持用户界面元素的物理尺寸（从 用户的视角），便可实现“密度独立性”。

保持密度独立性很重要，因为如果没有此功能，UI 元素（例如 按钮）在低密度屏幕上看起来较大，在高密度屏幕上看起来较小。这些 密度相关的大小变化可能给应用布局和易用性带来问题。图 2 和 3 分别显示了应用不提供密度独立性和 提供密度独立性时的差异。

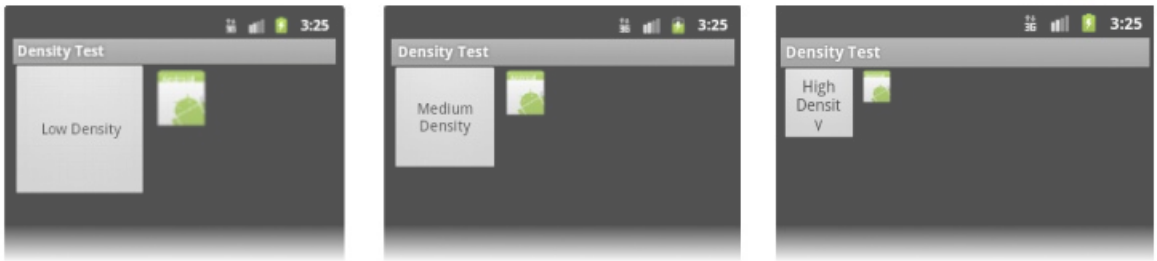

图 2. 不支持不同密度的示例应用在低、中、高密度屏幕上的显示情况。

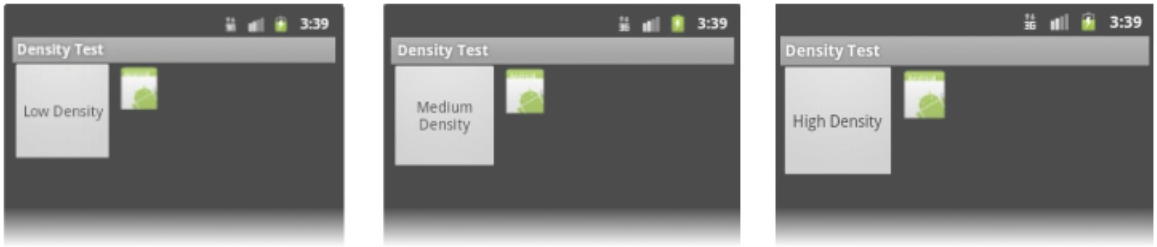

图 3. 良好支持不同密度（密度独立）的示例应用在低、中、高密度屏幕上的显示情况。

Android 系统可帮助您的应用以两种方式实现密度独立性：

- 系统根据当前屏幕密度扩展 dp 单位数
- 系统在必要时可根据当前屏幕 密度将可绘制对象资源扩展到适当的大小

在图 2 中，文本视图和位图可绘制对象具有以像素（`px` 单位）指定的尺寸，因此视图的物理尺寸在低密度屏幕上更大，在高密度 屏幕上更小。这是因为，虽然实际屏幕尺寸可能相同，但高密度屏幕 的每英寸像素更多（同样多的像素在一个更小的区域内）。在图 3 中，布局 尺寸以密度独立的像素（`dp` 单位）指定。由于 密度独立像素的基线是中密度屏幕，因此具有中密度屏幕的设备看起来 与图 2 一样。但对于低密度和高密度屏幕，系统 将分别增加和减少密度独立像素值，以适应 屏幕。

大多数情况下，确保应用中的屏幕独立性很简单，只需以适当的密度独立像素（`dp` 单位）或 `"wrap_content"` 指定所有 布局尺寸值。系统然后根据适用于当前屏幕密度的缩放比例适当地缩放位图可绘制对象，以 适当的大小显示。

但位图缩放可能导致模糊或像素化位图，您或许已经在上面的屏幕截图中 发现了这些问题。为避免这些伪影，应为 不同的密度提供替代的位图资源。例如，应为高密度 屏幕提供分辨率较高的位图，然后系统对中密度 屏幕将使用这些位图，而无需调整位图大小。下一节详细说明如何为不同的屏幕配置提供备用资源。

如何支持多种屏幕

Android 支持多种屏幕的基础是它能够管理针对当前屏幕配置 以适当方式渲染应用的布局和位图 可绘制对象。系统可处理大多数工作，通过适当地 缩放布局以适应屏幕尺寸/密度和根据屏幕密度缩放位图可绘制对象 ，在每种屏幕配置中渲染您的应用。但是，为了更适当地处理不同的屏幕配置 ，还应该：

- **在清单中显式声明您的应用 支持哪些屏幕尺寸**

  通过声明您的应用支持哪些屏幕尺寸，可确保只有 其屏幕受支持的设备才能下载您的应用。声明对 不同屏幕尺寸的支持也可影响系统如何在较大 屏幕上绘制您的应用 — 特别是，您的应用是否在屏幕兼容模式中运行。

  要声明应用支持的屏幕尺寸，应在清单文件中包含 `<supports-screens>` 元素。

- **为不同屏幕尺寸提供不同的布局**

  默认情况下，Android 会调整应用布局的大小以适应当前设备屏幕。大多数 情况下效果很好。但有时 UI 可能看起来不太好，需要针对 不同的屏幕尺寸进行调整。例如，在较大屏幕上，您可能要调整 某些元素的位置和大小，以利用其他屏幕空间，或者在较小屏幕上， 可能需要调整大小以使所有内容纳入屏幕。

  可用于提供尺寸特定资源的配置限定符包括 `small`、`normal`、`large` 和 `xlarge`。例如，超大屏幕的布局应使用 `layout-xlarge/`。

  从 Android 3.2（API 级别 13）开始，以上尺寸组已弃用，您 应改为使用 `sw<N>dp` 配置限定符来定义布局资源 可用的最小宽度。例如，如果多窗格平板电脑布局 需要至少 600dp 的屏幕宽度，应将其放在 `layout-sw600dp/` 中。声明适用于 Android 3.2 的平板电脑布局一节将进一步讨论如何使用新技术声明布局资源。

- **为不同屏幕密度提供不同的位图可绘制对象**

  默认情况下，Android 会缩放位图可绘制对象（`.png`、`.jpg` 和 `.gif` 文件）和九宫格可绘制对象（`.9.png` 文件），使它们以适当的 物理尺寸显示在每部设备上。例如，如果您的应用只为 基线中密度屏幕 (mdpi) 提供位图可绘制对象，则在高密度 屏幕上会增大位图，在低密度屏幕上会缩小位图。这种缩放可能在 位图中造成伪影。为确保位图的最佳显示效果，应针对 不同屏幕密度加入不同分辨率的替代版本。

  可用于密度特定资源的配置限定符（在下面详述） 包括 `ldpi`（低）、`mdpi`（中）、 `hdpi`（高）、`xhdpi`（超高）、`xxhdpi` （超超高）和 `xxxhdpi`（超超超高）。例如，高密度屏幕的位图应使用 `drawable-hdpi/`。

  > **注**：仅当要在 xxhdpi 设备上提供比正常位图大的启动器图标时才需要提供 `mipmap-xxxhdpi` 限定符。无需为所有应用的图像提供 xxxhdpi 资源。

有些设备会将启动器图标增大 25%。例如，如果您的最高 密度启动器图标已是超超高密度，缩放处理会降低其 清晰度。因此应在 `mipmap-xxxhdpi` 目录中提供更高密度的启动器图标，系统将改为增大较小 的图标。

请参阅提供 xxx-高密度启动器图标以了解详细信息。对启动程序图标以外的 UI 元素不应使用 `xxxhdpi` 限定符。

> **注**：将您的所有启动器图标放在 `res/mipmap-[density]/` 文件夹中，而非 `res/drawable-[density]/` 文件夹中。无论安装应用的设备屏幕分辨率如何，Android 系统都会将资源保留在这些密度特定的文件夹中，例如 mipmap-xxxhdpi。此 行为可让启动器应用为您的应用选择要显示在主 屏幕上的最佳分辨率图标。如需了解有关使用 mipmap 文件夹的详细信息，请参阅管理项目概览。

尺寸和密度配置限定符对应于 前面支持的屏幕范围中所述的通用尺寸和密度。

> **注**：如果不熟悉配置限定符以及 系统如何使用它们来应用备用资源，请参阅提供备用资源了解详细信息。

在运行时，系统通过 以下程序确保任何给定资源在当前屏幕上都能保持尽可能最佳的显示效果：

1. 系统使用适当的备用资源

   根据当前屏幕的尺寸和密度，系统将使用您的应用中提供的任何尺寸和 密度特定资源。例如，如果设备有 高密度屏幕，并且应用请求可绘制对象资源，系统将查找 与设备配置最匹配的可绘制对象资源目录。根据可用的其他 备用资源，包含 `hdpi` 限定符（例如 `drawable-hdpi/`）的资源目录可能是最佳匹配项，因此系统将使用此 目录中的可绘制对象资源。

2. 如果没有匹配的资源，系统将使用默认资源，并按需要向上 或向下扩展，以匹配当前的屏幕尺寸和密度。

   “默认”资源是指未标记配置限定符的资源。例如，`drawable/` 中的资源是默认可绘制资源。 系统假设默认资源设计用于基线屏幕尺寸和密度，即 正常屏幕尺寸和中密度。 因此，系统对于高密度屏幕向上扩展默认密度 资源，对于低密度屏幕向下扩展。

   当系统查找密度特定的资源但在 密度特定目录中未找到时，不一定会使用默认资源。系统在缩放时可能 改用其他密度特定资源提供更好的效果。例如，查找低密度资源但该资源不可用时， 系统会缩小资源的高密度版本，因为 系统可轻松以 0.5 为系数将高密度资源缩小至低密度资源，与以 0.75 为系数 缩小中密度资源相比，伪影更少。

如需有关 Android 如何通过使配置限定符与设备配置匹配来选择备用资源的更多信息，请参阅Android 如何查找最佳匹配资源。

使用配置限定符

Android 支持多种配置限定符，可让您控制系统 如何根据当前设备屏幕的特性选择备用资源。配置限定符是可以附加到 Android 项目中资源目录的字符串，用于指定在其中设计资源的配置。

要使用配置限定符：

1. 在项目的 `res/` 目录中新建一个目录，并使用以下 格式命名： `<resources_name>-<qualifier>`
   - `<resources_name>` 是标准资源名称（例如 `drawable` 或 `layout`）。
   - `<qualifier>` 是下表 1 中的配置限定符，用于指定 要使用这些资源的屏幕配置（例如 `hdpi` 或 `xlarge`）。

   您可以一次使用多个 `<qualifier>` — 只需使用短划线分隔每个 限定符。

2. 将适当的配置特定资源保存在此新目录下。这些资源 文件的名称必须与默认资源文件完全一样。

例如，`xlarge` 是超大屏幕的配置限定符。将 此字符串附加到资源目录名称（例如 `layout-xlarge`）时，它指向 要在具有超大屏幕的设备上使用这些资源的系统。

表 1. 可用于为 不同屏幕配置提供特殊资源的配置限定符。

屏幕特性	限定符	说明
尺寸	`small`	适用于*小*尺寸屏幕的资源。
	`normal`	适用于*正常*尺寸屏幕的资源。（这是基线尺寸。）
	`large`	适用于*大*尺寸屏幕的资源。
	`xlarge`	适用于*超大*尺寸屏幕的资源。
密	`ldpi`	适用于低密度 (*ldpi*) 屏幕 (~120dpi) 的资源。

度	mdpi	适用于中密度 (*mdpi*) 屏幕 (~160dpi) 的资源。（这是基线 密度。）
	hdpi	适用于高密度 (*hdpi*) 屏幕 (~240dpi) 的资源。
	xhdpi	适用于超高密度 (*xhdpi*) 屏幕 (~320dpi) 的资源。
	xxhdpi	适用于超超高密度 (*xxhdpi*) 屏幕 (~480dpi) 的资源。
	xxxhdpi	适用于超超超高密度 (*xxxhdpi*) 屏幕 (~640dpi) 的资源。此限定符仅适用于 启动器图标，请参阅上面的注。
	nodpi	适用于所有密度的资源。这些是密度独立的资源。不管当前屏幕的密度如何，系统都不会 缩放以此限定符标记的资源。
	tvdpi	适用于密度介于 mdpi 和 hdpi 之间屏幕（约为 213dpi）的资源。它并不是 “主要”密度组，主要用于电视，而大多数应用都不 需要它 — 对于大多数应用而言，提供 mdpi 和 hdpi 资源便已足够，系统将根据需要对其进行 缩放。如果发现必须提供 tvdpi 资源，应以 1.33*mdpi 的系数 调整其大小。例如，mdpi 屏幕的 100px x 100px 图像应该相当于 tvdpi 的 133px x 133px。
方向	land	适用于横屏（长宽比）的资源。
	port	适用于竖屏（高宽比）的资源。
纵横比	long	适用于纵横比明显高于或宽于（分别在竖屏 或横屏时）基线屏幕配置的屏幕的资源。
	notlong	适用于使用纵横比类似于基线屏幕 配置的屏幕的资源。

> **注：**如果是为 Android 3.2 和 更高版本开发应用，请参阅有关声明适用于 Android 3.2 的平板电脑布局的章节，了解 在为特定屏幕尺寸声明布局资源时应使用的 新配置限定符（而不是使用表 1 中的尺寸限定符）。

如需了解有关这些限定符如何粗略地对应于实际屏幕 尺寸和密度的更多信息，请参阅本文档前面的支持的屏幕范围 。

例如，以下应用资源目录 为不同屏幕尺寸和不同可绘制对象提供不同的布局设计。使用 `mipmap/` 文件夹放置 启动器图标。

```
res/layout/my_layout.xml               // layout for normal screen size ("default")
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/graphic.png          // bitmap for medium-density
res/drawable-hdpi/graphic.png          // bitmap for high-density
res/drawable-xhdpi/graphic.png         // bitmap for extra-high-density
res/drawable-xxhdpi/graphic.png        // bitmap for extra-extra-high-density

res/mipmap-mdpi/my_icon.png          // launcher icon for medium-density
res/mipmap-hdpi/my_icon.png          // launcher icon for high-density
res/mipmap-xhdpi/my_icon.png         // launcher icon for extra-high-density
res/mipmap-xxhdpi/my_icon.png        // launcher icon for extra-extra-high-density
res/mipmap-xxxhdpi/my_icon.png       // launcher icon for extra-extra-extra-high-density
```

如需了解如何使用备用资源的更多信息以及 配置限定符的完整列表（不只是屏幕配置），请参阅 提供备用资源。

请注意，当 Android 系统在运行时选择使用哪些资源时，它会使用 特定逻辑确定“最佳匹配”资源。也就是说，您使用的限定符无 需在所有情况下精确匹配当前屏幕配置，系统也可 使用它们。特别是，根据屏幕尺寸限定符选择资源时，如果没有更好的匹配资源，则系统将 使用专为小于当前屏幕的屏幕而设计的 资源（例如，如有必要，大尺寸屏幕将使用标准尺寸的屏幕 资源）。但是，如果唯一可用的资源大于当前屏幕， 则系统不会使用这些资源，并且如果没有其他资源与设备 配置匹配，应用将会崩溃（例如，如果所有布局资源均用 `xlarge` 限定符标记， 但设备是标准尺寸的屏幕）。如需有关系统如何选择资源的更多信息，请参阅Android 如何查找最佳匹配资源。

> **提示：**如果您有一些系统 应该永远不会缩放（或许是因为您在 运行时亲自对图像做一些调整）的可绘制对象资源，则应将它们放在有 `nodpi` 配置限定符的目录中。 使用此限定符的资源被视为与密度无关，系统不会缩放 它们。

设计替代布局和可绘制对象

您应该创建的备用资源类型取决于应用的需求。 通常，您应该使用尺寸和方向限定符提供替代布局资源 ，并且使用密度限定符提供替代位图可绘制对象资源。

以下各节摘要说明您可能要如何使用尺寸和密度限定符 来分别提供替代布局和可绘制对象。

替代布局

一般而言，在不同的屏幕配置上测试应用后，您会知道 是否需要用于不同屏幕尺寸的替代布局。例如：

- 在小屏幕上测试时，可能会发现您的布局不太适合 屏幕。例如，小屏幕设备的屏幕宽度可能无法容纳一排 按钮。在此情况下，您应该为小屏幕提供调整 按钮大小或位置的替代布局。
- 在超大屏幕上测试时，可能会发现您的布局无法 有效地利用大屏幕，并且明显拉伸填满屏幕。 在此情况下，您应该为超大屏幕提供替代布局，以提供 针对大屏幕（例如平板电脑）优化、重新设计的 UI。

  虽然您的应用不使用替代布局也能在大屏幕上正常运行，但 必须让用户感觉您的应用看起来像是专为其 设备而设计。如果 UI 明显拉伸，用户很可能对 应用体验不满意。
- 而且，对比横屏测试和竖屏测试时 可能会发现，竖屏时置于底部的 UI 在横屏时应位于屏幕右侧。

简而言之，您应确保应用布局：

- 适应小屏幕（让用户能实际使用您的应用）
- 已针对大屏幕优化，可以利用其他屏幕空间
- 已同时针对横屏和竖屏方向优化

如果 UI 使用的位图即使在系统缩放 布局后也需要适应视图大小（例如按钮的背景图片），则应使用九宫格位图文件。九宫格文件基本上是一个指定可拉伸的二维区域的 PNG 文件。 当系统需要缩放使用位图的视图时，系统 会拉伸九宫格位图，但只拉伸指定的区域。因此，您无 需为不同的屏幕尺寸提供不同的可绘制对象，因为九宫格位图可 调整至任何大小。但您应该为不同的屏幕密度提供 九宫格文件的替代版本。

替代可绘制对象

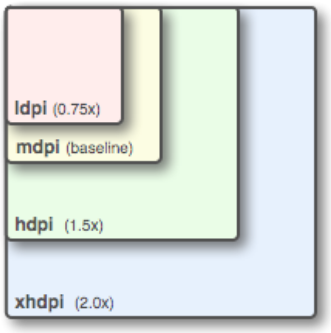

图 4. 支持每种密度的 位图可绘制对象的相对大小。

基本上每个应用都应该具有不同密度的替代可绘制对象 资源，因为基本上每个应用都有启动器图标，而且该图标应该在 所有屏幕密度中看起来都很好。同样，如果您的应用中包含其他位图可绘制对象（例如 应用中的菜单图标或其他图形），则应该为不同密度提供替代版本或 每种密度一个版本。

> **注：**您只需要为 位图文件（`.png`、`.jpg` 或 `.gif`）和九宫格文件 (`.9.png`) 提供密度特定的可绘制对象。如果您使用 XML 文件定义形状、颜色或其他可绘制对象资源，应该 将一个副本放在默认可绘制对象目录中 (`drawable/`)。

要为不同的密度创建替代位图可绘制对象，应遵循六种通用密度之间的 **3:4:6:8:12:16 缩放比率**。例如，如果您的 位图可绘制对象是对中密度屏幕使用 48x48 像素，则所有不同的尺寸应为：

- 36x36 (0.75x) 用于低密度
- 48x48（1.0x 基线）用于中密度
- 72x72 (1.5x) 用于高密度
- 96x96 (2.0x) 用于超高密度
- 144x144 (3.0x) 用于超超高密度
- 192x192 (4.0x) 用于超超超高密度（仅限启动器图标；请参阅上面的 注）

如需了解有关设计图标的更多信息，请参阅图标设计指南， 其中包含各种位图可绘制对象（例如启动器图标、菜单 图标、状态栏图标、选项卡

图标等）的大小信息。

声明适用于 Android 3.2 的平板电脑布局

对于第一代运行 Android 3.0 的平板电脑，声明平板电脑 的正确方式是将它们放在有 `xlarge` 配置限定符的目录（例如 `res/layout-xlarge/`）中。为适应其他类型的平板电脑和屏幕 尺寸 — 特别是 7 英寸平板电脑 — Android 3.2 引入了为更具体的屏幕尺寸指定资源 的新方式。新技术基于布局需要的空间量 （例如 600dp 宽），而不是尝试让您的布局容纳通用化的尺寸组 （例如*大*或*超大*）。

使用通用化的尺寸组时，为 7 英寸平板电脑设计很棘手的原因在于， 7 英寸平板电脑在技术上与 5 英寸手机属于同一个组（*大*组）。虽然 这两种设备在尺寸上似乎很接近，但用于 应用 UI 的空间量明显不同，用户交互的方式也是如此。因此，7 英寸和 5 英寸 屏幕不一定使用相同的布局。为便于您为这两种屏幕提供不同的 布局，Android 现在允许您 根据实际适用于应用布局的宽度和/或高度指定布局资源（ 以 dp 单位数指定）。

例如，在设计要用于平板电脑样式设备的布局之后，您可能 发现该布局在屏幕宽度小于 600dp 时不适用。此阈值 于是变成平板电脑布局需要的最小尺寸。因此，您现在可以指定应仅当至少有 600dp 宽度供应用的 UI 使用时才使用这些布局资源。

应选择一个宽度并将其设计为最小尺寸，或者在布局设计完成后测试 其支持的最小宽度。

> **注：**请记住，这些新尺寸 API 使用的所有数据是密度独立的像素 (dp) 值，您的布局尺寸也应始终 使用 dp 单位定义，因为您关注的是系统考虑屏幕密度后可用的屏幕空间量（与使用原始像素分辨率相反）。如需了解 密度独立像素的更多信息，请参阅本文档前面的术语和概念 。

使用新尺寸限定符

表 2 摘要列出了您可以根据 布局可用空间指定的不同资源配置。与传统的屏幕尺寸组（小、 正常、大和超大）相比，这些新的限定符可用于更多地控制 应用支持的屏幕尺寸。

> **注：**您使用这些限定符指定的尺寸 **不是实际屏幕尺寸**。更确切地说，尺寸是 **可用于 Activity 窗口**的宽度或高度（dp 单位）。Android 系统可能将某些屏幕用于系统 UI（例如屏幕底部的系统栏或 顶部的状态栏），因此有些屏幕可能不适用于您的布局。因此， 您声明的尺寸应与 Activity 需要的尺寸具体相关 — 系统 在声明向您的布局提供的空间量时会计算系统 UI 使用的任何空间。 另请注意，操作栏被视为 应用的窗口空间的一部分，但您的布局未声明此事，因此会减少 您的布局可用的空间，您在设计时必须考虑进去。

表 2. 屏幕尺寸的新配置限定符 （在 Android 3.2 中引入）。

屏幕配置	限定符值	说明

smallestWidth	`sw<N>dp`
	
	示例：
	`sw600dp`
	`sw720dp`
		屏幕的基本尺寸，由可用屏幕区域的最小尺寸指定。 具体来说，设备的 smallestWidth 是屏幕可用高度和宽度的最小尺寸（您也可以将其视为屏幕的“最小可能宽度”）。无论屏幕的当前方向如何，您均可使用此限定符确保应用 UI 的可用宽度至少为 `<N>`dp。
		
		例如，如果布局要求屏幕区域的最小尺寸始终至少为 600 dp，则可使用此限定符创建布局资源 `res/layout-sw600dp/`。仅当可用屏幕的最小尺寸至少为 600dp 时，系统才会使用这些资源，而不考虑 600dp 所代表的边是用户所认为的高度还是宽度。smallestWidth 是设备的固定屏幕尺寸特性；**设备的 smallestWidth 不会随屏幕方向的变化而改变**。
		
		设备的 smallestWidth 将屏幕装饰元素和系统 UI 考虑在内。例如，如果设备的屏幕上有一些永久性 UI 元素占据沿 smallestWidth 轴的空间，则系统会声明 smallestWidth 小于实际屏幕尺寸，因为这些屏幕像素不适用于您的 UI。
		
		这可替代通用化的屏幕尺寸限定符（小、正常、大、超大）， 可让您为 UI 可用的有效尺寸定义不连续的数值。 使用 smallestWidth 定义一般屏幕尺寸很有用，因为宽度 通常是设计布局时的驱动因素。UI 经常会垂直滚动，但 对其水平需要的最小空间具有非常硬性的限制。可用的宽度也是 确定是否对手机使用单窗格布局或是对平板电脑使用多窗格布局 的关键因素。因此，您可能最关注每部 设备上的最小可能宽度。

可用屏幕宽度	`w<N>dp`
	
	示例：
	`w720dp`
	`w1024dp`
		指定资源应该使用的最小可用宽度（dp 单位） — 由 `<N>` 值定义。当屏幕的方向在横屏与竖屏之间切换时，系统对应的 宽度值将会变化，以 反映 UI 可用的当前实际宽度。
		
		这对于确定是否使用多窗格布局往往很有用，因为即使是在 平板电脑设备上，您也通常不希望竖屏像横屏一样 使用多窗格布局。因此，您可以使用此功能指定布局需要的最小宽度，而 无需同时使用屏幕尺寸和方

		向限定符。

可用屏幕高度	`h<N>dp`
	
	示例：
	`h720dp`
	`h1024dp`
	等等
		指定资源应该使用的最小屏幕高度（dp 单位） — 由 `<N>` 值定义。当屏幕的方向在横屏与竖屏之间切换时，系统 对应的高度值将会变化，以 反映 UI 可用的当前实际高度。
		
		使用此方式定义 布局需要的高度很有用，它与使用 `w<N>dp` 定义 所需宽度的方式相同，无需同时使用屏幕尺寸和方向限定符。 但大多数应用不需要此限定符，考虑到 UI 经常垂直滚动， 因此高度更弹性，而宽度更刚性。

虽然使用这些限定符似乎比使用屏幕尺寸组更复杂，但 当您确定 UI 的要求后，它实际上应该更简单。在设计 UI 时， 您主要关注的可能是应用在 手机样式 UI 与使用多窗格的平板电脑样式 UI 之间切换时的实际尺寸。此确切的精确时间 取决于特定设计 — 可能平板电脑布局需要 720dp 宽度， 但 600dp、480dp 或这两者之间的某个值就够了。使用表 2 中的这些限定符 可以控制布局切换时的精确尺寸。

如需有关这些尺寸配置限定符的更多讨论，请参阅 提供资源文档。

配置示例

为帮助您针对不同的设备类型确定某些设计，下面提供了一些 常见的屏幕宽度值：

- 320dp：常见手机屏幕（240x320 ldpi、320x480 mdpi、480x800 hdpi 等）。
- 480dp：中间平板电脑，例如 Streak (480x800 mdpi)。
- 600dp：7 英寸平板电脑 (600x1024 mdpi)。
- 720dp：10 英寸平板电脑（720x1280 mdpi、800x1280 mdpi 等）。

利用表 2 中的尺寸限定符，您的应用可以使用要用于宽度和/或高度的 的任何值，在用于手机和平板电脑的不同布局资源之间切换。例如， 如果 600dp 是平板电脑布局支持的最小可用宽度，您可以提供以下两 组布局：

```
res/layout/main_activity.xml           # For handsets
res/layout-sw600dp/main_activity.xml   # For tablets
```

在此情况下，可用屏幕空间的最小宽度必须是 600dp，才可 应用平板电脑布局。

对于要进一步自定义 UI 以区分不同尺寸 （例如 7 英寸和 10 英寸平板电脑）的其他情况，您可以定义其他最小宽度布局：

```
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)
```

请注意，上面两组示例资源使用“最小宽度”限定符 `sw<N>dp`，用于指定屏幕两边的最小值，而不管设备 当前的方向如何。因此，使用 `sw<N>dp` 是指定 布局可用于整体屏幕尺寸的简便方法，它会忽略屏幕的方向。

但在某些情况下，可能 必须确定布局 *当前*可用的精确宽度或高度。例 如，如果是并排显示两个片段的双窗格布局，则只要 屏幕提供至少 600dp 的宽度（无论设备是横屏还是竖屏）， 您可能就要使用该布局。在此情况下，您的资源可能与以下所示类似：

```
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，一部设备可能实际使用两种布局，具体取决于屏幕的方向（如果 可用的宽度在一个方向上至少为 600dp，而在另一个方向上小于 600dp）。

如果您关注可用高度，便可使用 `h<N>dp` 限定符执行同样的操作。或者，如果您需要很具体，甚至可以结合 `w<N>dp` 与 `h<N>dp` 限定符。

声明屏幕尺寸支持

在对不同的屏幕尺寸实现您的布局后，在 清单文件中声明您的应用支持哪些屏幕相当重要。

与用于屏幕尺寸的新配置限定符一起，Android 3.2 为 <supports-screens> 清单元素引入了新的属性：

`android:requiresSmallestWidthDp`

> 指定需要的最小 smallestWidth。smallestWidth 是必须为您的应用 UI 提供的 屏幕空间的最短尺寸（dp 单位）—即 可用屏幕的两个尺寸中的最短者。因此，为使设备 与您的应用兼容，设备的 smallestWidth 必须等于或大于此 值。（通常，无论屏幕的当前方向如何， 此值都是布局支持的“最小宽度”。）

例如，如果您的应用只用于最小可用宽度为 600dp 的平板电脑样式设备：

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

但是，如果您的应用支持 Android 支持的所有屏幕尺寸（小至 426dp x 320dp），则无需声明此属性，因为应用 需要的最小宽度就是任何设备上可以实现的最小宽度。

> **注意：**Android 系统不关注此 属性，因为它不影响应用在运行时的行为，而是被用于 在服务（例如 Google Play）上过滤您的应用。但是， **Google Play 目前不支持此属性用于过滤**（在 Android 3.2 上），因此如果您的应用不支持 小屏幕，您应继续使用其他尺寸属性。

`android:compatibleWidthLimitDp`

此属性可让您指定用户支持的最大“最小宽度”，将屏幕兼容性模式用作 用户可选的功能 。如果设备可用屏幕的最小边大于您在这里的值， 用户仍可安装您的应用，但提议在屏幕兼容性模式下运行。默认 情况下，屏幕兼容性模式会停用，并且您的布局照例会调整大小以适应屏幕，但按钮会显示在系统栏中，可让用户打开和关闭屏幕兼容性 模式。

> **注：**如果您的应用可针对大 屏幕正确调整大小，则无需使用此属性。建议不要使用此 属性，而是按照本文档的 建议，确保您的布局针对较大屏幕调整大小。

`android:largestWidthLimitDp`

此属性可让您指定应用支持的最大“最小宽度”来强制启用屏幕兼容性模式。 如果设备可用屏幕的最小 边大于您在这里的值，应用将在屏幕 兼容性模式下运行，且用户无法停用该模式。

> **注：**如果您的应用可针对大 屏幕正确调整大小，则无需使用此属性。建议不要使用此 属性，而是按照本文档的 建议，确保您的布局针对较大屏幕调整大小。

> **注意：**针对 Android 3.2 及更高版本开发时，您 应改为将旧屏幕尺寸属性与上列 属性结合使用。同时使用新属性和旧尺寸属性可能导致 非预期的行为。

如需了解每个属性的更多信息，请跟随上面各自的链接。

最佳做法

支持多种屏幕的目标是创建一款在 Android 系统支持的通用屏幕尺寸上都可以 正常运行且显示良好的应用。本文档 前面各节内容介绍了 Android 系统如何使您的 应用适应屏幕配置，以及如何在不同的 屏幕配置上自定义应用的外观。本节提供另外一些提示以及有助于 确保应用针对不同屏幕配置正确缩放的 技巧概览。

下面是有关如何确保应用在 不同屏幕上正常显示的快速检查清单：

1. 在 XML 布局文件中指定尺寸时使用 `wrap_content`、`match_parent` 或 `dp` 单位 。
2. 不要在应用代码中使用硬编码的像素值
3. 不要使用 `AbsoluteLayout`（已弃用）
4. 为不同屏幕密度提供替代位图可绘制对象

下文将提供更详细的信息。

1. 对布局尺寸使用 wrap_content、match_parent 或 dp 单位

为 XML 布局文件中的视图定义 `android:layout_width` 和 `android:layout_height` 时，使用 `"wrap_content"`、 `"match_parent"` 或 `dp` 单位可确保在当前设备屏幕上为 视图提供适当的尺寸。

例如，`layout_width="100dp"` 的视图在 中密度屏幕上测出宽度为 100 像素，在高密度屏幕上系统会将其扩展至 150 像素宽， 因此视图在屏幕上占用的物理空间大约相同。

类似地，您应选择 `sp`（缩放独立的像素）来定义文本 大小。`sp` 缩放系数取决于用户设置，系统 会像处理 `dp` 一样缩放大小。

2. 不要在应用代码中使用硬编码的像素值

由于性能的原因和简化代码的需要，Android 系统使用像素作为 表示尺寸或坐标值的标准单位。这意味着， 视图的尺寸在代码中始终以像素表示，但始终基于当前的屏幕密度。 例如，如果 `myView.getWidth()` 返回 10，则表示视图在 当前屏幕上为 10 像素宽，但在更高密度的屏幕上，返回的值可能是 15。如果 在应用代码中使用像素值来处理预先未针对 当前屏幕密度缩放的位图，您可能需要缩放代码中使用的像素值，以与 未缩放的位图来源匹配。

如果应用在运行时操作位图或处理像素值，请参阅 下面有关其他密度注意事项的一节。

3. 不要使用 AbsoluteLayout

与其他布局小工具不同，`AbsoluteLayout` 会强制 使用固定位置放置其子视图，很容易导致 在不同显示屏上显示效果不好的用户界面。因此，`AbsoluteLayout` 在 Android 1.5（API 级别 3）上便已弃用。

您应改用 `RelativeLayout`，它会使用相对定位 来放置其子视图。例如，您可以指定按钮小部件显示在文本小工具的“右边”。

4. 使用尺寸和密度特定资源

虽然系统会根据当前屏幕 配置扩展布局，但您在不同的屏幕尺寸上可能要调整 UI，以及提供 针对不同密度优化的可绘制对象。这基本上是重申 本文档前面的信息。

如果需要精确控制应用在不同 屏幕配置上的外观，请在配置特定的 资源目录中调整您的布局和位图可绘制对象。例如，考虑要显示在 中密度和高密度屏幕上的图标。只需创建两种不同大小的图标 （例如中密度使用 100x100，高密度使用 150x150），然后使用适当的限定符 以适当的方向放置两个 变体：

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

> **注：**如果密度限定符在目录名称中未定义， 系统会假设该目录中的资源是针对基线中 密度而设计，对于其他密度将会适当地缩放。

如需了解有效配置限定符的更多信息，请参阅本文档前面的使用 配置限定符。

其他密度注意事项

本节详细说明 Android 如何在不同 屏幕密度上对位图可绘制对象执行缩放，以及如何进一步控制在不同密度屏幕上位图的绘制。 本节信息对大多数应用应该不怎么重要，除非您的 应用在不同屏幕密度上运行或 操控图形时遇到了问题。

为更好地了解在运行时 操控图形时如何支持多种密度，您应该先了解，系统通过以下方式帮助确保正确 缩放位图：

1. *资源（例如位图可绘制对象）的预缩放*

   根据当前屏幕的密度，系统将使用您的应用中提供的任何尺寸或 密度特定资源，并且不加缩放而显示它们。如果没有可用于正确密度 的资源，系统将加载默认资源，并按需要向上或向下扩展，以 匹配当前屏幕的密度。系统假设默认资源（ 没有配置限定符的目录中的资源）针对基线屏幕密度 (mdpi) 而设计， 除非它们加载自密度特定的资源目录。因此，系统 会执行预缩放，以将位图调整至适应当前屏幕 密度的大小。

   如果您请求预缩放的资源的尺寸，系统将返回 代表缩放*后*尺寸的值。例如，针对 mdpi 屏幕以 50x50 像素 设计的位图在 hdpi 屏幕上将扩展至 75x75 像素（如果没有 用于 hdpi 的备用资源），并且系统会这样报告大小。

   有时您可能不希望 Android 预缩放 资源。避免预缩放最简单的方法是将资源放在 有 `nodpi` 配置限定符的资源目录中。例如：

   ```
   res/drawable-nodpi/icon.png
   ```

   当系统使用此文件夹中的 `icon.png` 位图时，不会 根据当前设备密度缩放。

2. *像素尺寸和坐标的自动缩放*

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

   例如，假设设备具有 480x800 的 WVGA 高密度屏幕，大约 与传统 HVGA 屏幕的尺寸一样，但它运行的应用停用了 预缩放。在此情况下，系统在查询屏幕尺寸时会对应用“撒谎”，报告 320x533（屏幕密度的近似 mdpi 转换值）。然后，当 应用执行绘制操作时，例如作废从

(10,10) 到 (100, 100) 的矩形，系统会将它们缩放适当的量以转变坐标，并且实际 作废区域 (15,15) 到 (150, 150)。如果应用直接操控缩放的位图， 此差异可能会导致非预期的行为，但这被视为 确保应用最佳性能所需的合理权衡。如果遇到此 情况，请参阅将 dp 单位转换为像素 单位一节。

通常，**不应停用预缩放**。支持多种 屏幕的最佳方法是采用上面如何支持 多种屏幕中所述的基本技术。

如果您的应用操控位图或以某种其他方式直接与 屏幕上的像素交互，您可能需要采取其他步骤支持不同的屏幕密度。例 如，如果您通过计算手指滑过的像素数 来响应触控手势，则需使用适当的密度独立像素值，而不是实际 像素。

缩放运行时创建的位图对象

图 5. 预缩放的位图与自动缩放的 位图比较。

如果您的应用创建内存中位图（`Bitmap` 对象）， 系统在默认情况下假设位图是针对基线中密度屏幕而设计，然后 在绘制时自动缩放位图。当位图具有不明的密度属性时，系统会对 `Bitmap` 应用“自动缩放”。如果未正确 考虑当前设备的屏幕密度和指定位图的密度属性， 自动缩放可能导致缩放伪影，就像未提供备用资源一样。

要控制是否缩放运行时创建的 `Bitmap`，可以 使用 `setDensity()` 指定位图的密度， 从 `DisplayMetrics` 传递密度常量，例如 `DENSITY_HIGH` 或 `DENSITY_LOW`。

如果使用 `BitmapFactory` 创建 `Bitmap`，例如从文件或流创建，可以使用 `BitmapFactory.Options` 定义位图的属性（因为 它已经存在），确定系统是否要缩放或者如何缩放。例如，您可以使用 `inDensity` 字段定义 位图设计时的密度，使用 `inScaled` 字段指定位图是否应缩放以匹配当前设备的屏幕密度。

如果将 `inScaled` 字段设置为 `false`，然后停用 系统可能应用到位图的预缩放，则系统在绘制时将自动 缩放它。使用自动缩放代替预缩放可能耗用的 CPU 更多，但耗用的内存 更少。

图 5 所示为在高密度屏幕上加载低 (120)、中 (160) 和高 (240) 密度位图时预缩放和自动缩放机制产生的效果。差异 很小，因为所有位图都针对当前屏幕密度而缩放，但根据在绘制时 是预缩放还是自动缩放， 缩放后位图的外观略有不同。

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

将 dp 单位转换为像素单位

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

(240dpi) 的设备上，用户必须移动 16 pixels / 240 dpi（等于 一英寸的 1/15 或 1.7 毫米）。此距离更短，应用因此 似乎对用户更灵敏。

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

```
// The gesture threshold expressed in dp
private static final float GESTURE_THRESHOLD_DP = 16.0f;

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

但是，不能为此类事件定义任意阈值，而应 使用 ViewConfiguration 中的预缩放配置值。.

使用预缩放的配置值

您可以使用 ViewConfiguration 类访问 Android 系统使用的通常距离、 速度和时间。例如， 使用 getScaledTouchSlop() 可获取框架用作滚动阈值的距离（像素）：

```
private static final int GESTURE_THRESHOLD_DP = ViewConfiguration.get(myContext).getScaledTouchSlop();
```

ViewConfiguration 中以 getScaled 前缀 开头的方法确定会返回不管当前屏幕密度为何都会正常显示的 像素值。

如何在多个屏幕上测试您的应用

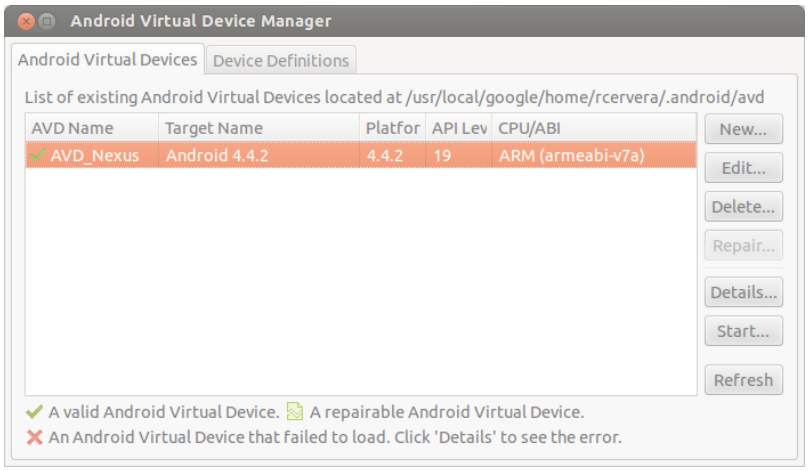

图 6. 一组用于测试屏幕支持的 AVD。

在发布应用之前，应在所有支持的屏幕 尺寸和密度中全面测试。Android SDK 包含可以使用的模拟器皮肤， 它们会复制您的应用可能要在其中运行的常见屏幕配置的 尺寸和密度。也可修改模拟器皮肤的默认尺寸、密度和分辨率， 以复制任何特定屏幕的特性。使用模拟器皮肤和其他自定义配置可测试任何可能的屏幕配置，因此您无 需仅仅为了测试应用的屏幕支持而购买不同的设备。

要设置环境以测试应用的屏幕支持，应使用能模拟您希望应用支持的 屏幕尺寸和密度的模拟器皮肤和屏幕配置创建 一系列 AVD (Android Virtual Devices)。要执行此操作，可以使用 AVD Manager 创建 AVD 并使用图形界面启动它们。

要启动 Android SDK Manager，从您的 Android SDK 目录执行 SDK Manager.exe（仅在 Windows 上），或者从 <sdk>/tools/ 目录执行 android（在所有平台上）。图 6 所示为用于测试不同屏幕配置的 AVD Manager（选择了 AVD）。

表 3 所示为 Android SDK 中可用的各种模拟器皮肤，可用 以模拟某些最常见的屏幕配置。

如需了解有关创建和使用 AVD 测试应用的详细信息，请参阅使用 AVD Manager 管理 AVD。

表 3. Android SDK（粗体表示）及其他 代表性解决方案中模拟器皮肤提供的 各种屏幕配置

	低密度 (120)，*ldpi*	中密度 (160)，*mdpi*	高密度 (240)，*hdpi*	超高密度 (320)，*xhdpi*
小屏幕	**QVGA (240x320)**		480x640	
正常屏幕	**WQVGA400 (240x400)** **WQVGA432 (240x432)**	**HVGA (320x480)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024	640x960
大屏幕	**WVGA800** (480x800)** **WVGA854** (480x854)**	**WVGA800* (480x800)** **WVGA854* (480x854)** 600x1024		
超大屏幕	1024x600	**WXGA (1280x800)**† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

* 要模拟此配置，在 创建使用 WVGA800 或 WVGA854 皮肤的 AVD 时请指定自定义密度 160。

** 要模拟此配置，在创建 使用 WVGA800 或 WVGA854 皮肤的 AVD 时请指定自定义密度 120。

† 此皮肤可用于 Android 3.0 平台

要查看支持任何指定屏幕配置的活动设备的相对数量，请参阅 屏幕尺寸和密度 仪表板。

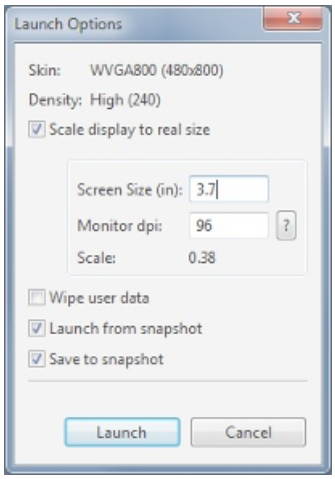

图 7. 从 AVD Manager 启动 AVD 时可以设置的尺寸和密度选项。

我们还建议您在 设置为以接近实际设备的物理尺寸运行的模拟器中测试应用。这样 更容易比较不同尺寸和密度时的结果。要 完成此操作，需要知道计算机显示器的大约密度 (dpi)， 例如 30 英寸 Dell 显示器的密度约为 96 dpi。从 AVD Manager 启动 AVD 时，可在 Launch Options 中指定用于模拟器和您的 显示器的屏幕尺寸 (dpi)，如图 7 所示。

如果要在使用内置皮肤 不支持的分辨率或密度的屏幕上测试应用，可以创建使用自定义分辨率或密度的 AVD。从 AVD Manager 创建 AVD 时，指定 Resolution， 而不要选择 Built-in Skin。

从命令行启动 AVD 时，可以使用 `-scale` 选项指定用于 模拟器的缩放比例。例如：

```
emulator -avd <avd_name> -scale 96dpi
```

要调整模拟器的大小，可使用 `-scale` 选项指定代表所需缩放系数的 0.1 至 3。

如需了解从命令行创建 AVD 的更多信息，请参阅从命令行管理 AVD。