Android - Cards de Referência

Drawables

Bem vindo ao nosso quinto card sobre Android! :-) Desta vez, vamos conversar sobre os Drawables. Embora pareça simples, saiba que existem diversas formas de você usar imagens em seu projeto. Não é só copiar e colar um PNG, tem mais detalhes que vão lhe ajudar muito!

DRAWABLES

Vamos ver agora os recursos que chamamos de drawable. Basicamente, podemos usar imagens PNG, JPG, GIF e um tipo especial chamado 9-Patch. Contudo, as imagens GIF não são indicadas. Evite usar elas.

As imagens ficam nas pastas **/res/drawable-***. Por que este "-*"? Simples, normalmente dividimos os drawables conforme o tamanho de tela e DPI, basicamente. Observe que quando você cria um projeto Android no Eclipse, já vem com as pastas drawable-ldpi, drawable-mdpi, drawable-hdpi e drawable-xhdpi.

A depender das configurações do dispositivo, será usada a imagem de uma pasta específica. Você referencia essas imagens em seu código Java através das constantes em **R.drawable.***.

Tem mais um detalhe, imagens não são apenas arquivos PNG, JPG ou GIF. Podemos ir um pouco além. Por exemplo, podemos definir um drawable como uma sequência de imagens. Ou um drawable pode ser uma lista de imagens que muda conforme determinados estados. Difícil? Na verdade, é fácil. E interessante.

BITMAP EM XML

Esta forma de definir um drawable é interessante, pois ele referencia uma imagem real, mas também permite definir atributos extras.

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
   android:src="@drawable/icon"
   android:tileMode="repeat" />
```

Observe o exemplo, retirado da documentação do Android. Vamos chamar este arquivo de **/res/drawable/icone.xml**. Referenciamos uma imagem real, mas definimos que ela irá preencher todo o espaço disponível fazendo uma repetição. Você pode usar os seguintes atributos na tag <bitmap>:

android:antialias para ativar ou desativar o antialiasing.

android:dither para ativar ou desativar o dithering.

android:filter para ativar ou desativar a filtragem de bitmap.

android:gravity para definir o posicionamento desta imagem de acordo com o container que a contém. As opções disponíveis são: top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, fill, clip_vertical, clip_horizontal.

A maioria destas opções são auto-explicativas. Vamos explicar algumas delas apenas. As opções que possuem fill_algo vão esticar a imagem para caber no espaço disponível. As opções clip_algo vão cortar a imagem nas bordas do container.

android:tileMode para definir se terá e como será a repetição do bitmap caso haja mais espaço para ocupar do que a largura e altura natural dele. Pode assumir os valores da tabela abaixo.

Valor	Descrição
disabled	Desabilitado. Não repete a imagem.
clamp	Replica a cor da borda se a imagem for desenhada fora dos limites.
repeat	Repete a imagem verticalmente e horizontalmente.
mirror	Repete a imagem verticalmente e horizontalmente, alternando com imagens espelhadas.

BITMAPS SOBREPOSTOS

Neste tipo, você tem uma lista de imagens que serão exibidas sequencialmente, iniciando pela primeira definida até a última. A última da lista é a exibida no topo de todas. Veja a imagem abaixo, retirada do próprio guia de referência do Android.

A forma de fazer isto é fácil. Primeiro, você precisa ter as três imagens em alguma pasta drawable. Depois, criar um arquivo XML em alguma pasta drawable também (**/res/drawable/layers.xml**). Agora, vamos usar a tag <layer-list>.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list>
   <item>
    <bitmap android:src="@drawable/android_red"
      android:gravity="center" />
   </item>
   <item android:top="10dp" android:left="10dp">
    <bitmap android:src="@drawable/android_green"
      android:gravity="center" />
   </item>
   <item android:top="20dp" android:left="20dp">
    <bitmap android:src="@drawable/android_blue"
      android:gravity="center" />
   </item>
</layer-list>
```

O código acima define exatamente o drawable da imagem anterior. Observe o atributo **android:top**, que é crescente em cada um. Faz sentido, pois é para cada um ficar logo abaixo do outro. Observe que o último elemento da lista, o android azul, ficou no topo de todos também. O código abaixo usa esse layer-list.

```
<ImageView
   android:src="@drawable/layers" />
```

IMAGENS POR ESTADO

Você também pode criar um arquivo XML que define um conjunto de imagens, mas cada uma será usada conforme determinados estados. Lembra que fizemos algo parecido com as cores? Aqui é a mesma coisa. Vamos ver um exemplo.

Primeiro, precisamos criar um arquivo **/res/drawable/imagem.xml**. Agora, vamos preencher ele com o conteúdo do trecho a seguir.

```
<?xml version="1.0" encoding="utf-8"?>
<selector>
   <item android:state_pressed="true"
       android:drawable="@drawable/pressionado" />
   <item android:state_focused="true"
       android:drawable="@drawable/focado" />
   <item android:state_hovered="true"
       android:drawable="@drawable/hovered" />
   <item android:drawable="@drawable/normal" />
</selector>
```

Estamos definindo uma imagem para cada estado. Quais são esses estados? São eles: state_pressed, state_focused, state_hovered, state_selected, state_checkable, state_checked, state_enabled, state_activated, state_window_focused.

Isto é interessante para o caso de você criar seus próprios botões usando imagens. Para cada situação, você define uma imagem.

LISTA DE NÍVEIS

Este tipo de drawable é bastante interessante para você fazer indicadores de carga, por exemplo. Você pode definir imagens e associar níveis a cada uma dela. Uma vez feito isto, pode em seu código Java, na Activity, dinamicamente mudar qual imagem será exibida conforme este nível.

Considere a listagem de código abaixo. Ela deve ser criada no arquivo **/res/drawable/bateria_drawable**.

```
<level-list>
   <item android:maxLevel="0"  android:maxLevel="25"
       android:drawable="@drawable/bateria_25" />
   <item android:maxLevel="25"  android:maxLevel="50"
       android:drawable="@drawable/bateria_50" />
   <item android:maxLevel="50"  android:maxLevel="75"
       android:drawable="@drawable/bateria_75" />
   <item android:maxLevel="75"  android:maxLevel="100"
       android:drawable="@drawable/bateria_100" />
</level-list>
```

Definimos quatro imagens e informamos o nível mínimo e máximo de cada uma. Agora, na Activity, queremos exibir uma imagem ou outro, parecendo os níveis de carga de uma bateria de celular.

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/imageView1"
    android:src="@drawable/bateria_drawable"/>
```

Acima, definimos ela como fonte de imagem para um ImageView. Em nossa Activity, vamos agora trocar essa imagem definindo o imageLevel dela.

```
public class MainActivity extends Activity {
        private ImageView imageView;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    imageView = (ImageView) findViewById(R.id.imageView1);
    imageView.setImageLevel(26);
  }
}
```

Observe que será exibida a imagem bateria_50. Concorda? O número 26 está entre o minLevel=25 e maxLevel=50. :-)

DRAWABLES EM TRANSIÇÃO

Você também pode definir duas imagens e uma transição cross-fade entre elas. Como nos demais passos, é necessário criar um arquivo XML dentro da pasta **/res/drawable/** com o nome **transicao.xml**. Veja o exemplo abaixo.

```
<transition>

  <item android:drawable="@drawable/imagem_1"/>

  <item android:drawable="@drawable/imagem_2"/>

</transition>
```

Temos duas imagens, onde haverá uma transição da primeira para a segunda. Como usar ela?

```
<ImageButton
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/imageButton1"
    android:src="@drawable/transicao"/>
```

No código Java da Activity, você pode definir o delay da transição conforme o código a seguir.

```
public class MainActivity extends Activity {
        private ImageButton imageButton;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    imageButton = (ImageButton)
findViewById(R.id.imageButton1);
    imageButton.startTransition(500);
  }
}
```

CLIP DRAWABLE

Você já viu essas barras de progresso que muita gente usa no Android? Quer fazer uma especial pra sua aplicação? É fácil. Você precisa apenas usar um Clip Drawable. Com ele, você define uma imagem e uma determinada área que será usada para exibir partes dessa imagem.

```
<clip android:drawable="@drawable/android"

  android:clipOrientation="horizontal"

  android:gravity="left" />
```

No exemplo acima, definimos que a imagem será uma chamada **android.png**, que o clip será feito de forma horizontal e começa da esquerda. :-) O efeito será igual ao da imagem abaixo.

Agora, precisamos definir qual objeto conterá a imagem e programaticamente aumentar o tamanho do clipping. Primeiro, precisamos de um ImageView.

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/imageView1"
    android:src="@drawable/clip"/>
```

Vamos fazer a animação no próximo trecho de código.

```
public class MainActivity extends Activity {
        private ImageView imageView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      imageView = (ImageView) findViewById(R.id.imageView1);
      ClipDrawable drawable = (ClipDrawable)
imageView.getDrawable();
      drawable.setLevel(drawable.getLevel()+1000);
   }
}
```

SHAPE DRAWABLES

Mais uma forma de criar imagens é através de shapes. Trocando em miúdos, é você criar imagens com as formas básicas, como retângulo, círculos... E é super simples fazer isto. Vamos ver alguns exemplos. Mais uma vez, você precisa criar um arquivo XML em **/res/drawable** para cada imagem. Vamos criar um **quadrado.xml** lá.

```
<shape

   android:shape="rectangle">

   <corners android:radius="8dp" />

</shape>
```

Como usar este novo "shape" que criamos? Você pode colocar ele como background de EditText, por exemplo, e personalizar todos os campos de texto de sua aplicação.

```
<TextView

   android:background="@drawable/quadrado"

   android:layout_height="wrap_content"

   android:layout_width="wrap_content" />
```

Aqui você tem muitas opções de tags. Vamos ver cada uma. Inicialmente, temos a tag <shape> e alguns atributos interessantes.

android:shape - define o tipo de shape que queremos. Pode assumir os valores rectangle, oval, line e ring.

Os atributos abaixo só fazem sentido se você informar como shape o "ring".

android:innerRadius - define o raio interno do anel.
android:innerRadiusRatio - define o raio interno do anel, expressa como uma razão entre a largura do anel.
android:thicknessRatio - espessura do anel, expressa como uma razão entre a largura do anel.
android:useLevel - deve ser verdadeiro para ser usado como um List Drawable.

Vamos a mais uma tag. Agora a <corners>, que é usada para definir propriedades dos cantos do shape. É muito usado em retângulos para criar bordas arredondadas. Pode ter as propriedades a seguir.

android:radius - define o raio para todas as bordas.
android:topLeftRadius - define o raio para a borda superior esquerda.
android:topRightRadius - define o raio para a borda superior direita.
android:bottomLeftRadius - define o raio para a borda inferior esquerda.
android:bottomRightRadius - define o raio para a borda inferior direita.

A próxima tag permite você definir um gradiente que ocupa o espaço interno da forma que você escolher. Trata-se da tag <gradient>, vamos ver seus atributos.

android:angle - define o ângulo para o gradiente, em graus. O valor zero é da esquerda para a direita. 90 é do topo para baixo. Deve ser sempre um valor múltiplo de 45.
android:centerX - posição no eixo X para o centro do gradiente.
android:centerY - posição no eixo Y para o centro do gradiente.
android:endColor - a cor final do gradiente.
android:startColor - a cor inicial do gradiente.
android:gradientRadius - o raio para o gradiente.
android:type - tipo de gradiente: linear, radial ou sweep.
android:useLevel - deve ser verdadeiro para ser usado como um List Drawable.

A próxima tag, chamada de <padding>, é para você definir o espaçamento do shape com relação a View que o contém. Alguns também traduzem padding para "enchimento", mas eu acho estranho. :-) Os atributos são óbvios, você vai informar o espaço para o topo, baixo, esquerda e direita usando **android:(top | bottom | left | right)**.

Mais uma tag, agora a <size> para você informar o tamanho do shape. Aqui, também é fácil, basta usar os atributos **android:(width | height)**.

A penúltima tag é a <solid>, com a qual você informa uma cor para preencher a forma que você está criando. Tem só um atributo, chamado **android:color**.

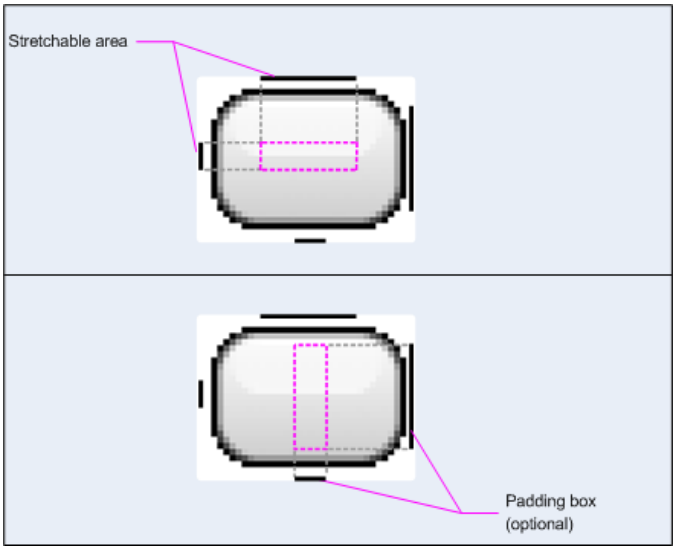
Agora a última, chamada <stroke>. Nela você define propriedades da linha que contorna a forma. As propriedades são:
android:width - largura da linha.
android:color - cor da linha.

android:dashGap - distância entre os traços que formam a linha.
android:dashWidth - o tamanho de cada traço que forma a linha. Faz sentido apenas se dashGap for definido.

9 Patch

Uma imagem em 9 Patch não é nenhum bicho de sete (ou nove?) cabeças, como muita gente pensa. É apenas uma imagem PNG normal, na qual você define áreas que podem ser esticadas ou não.

Pra que serve isso? Serve bastante para você aplicar em Views que precisam esticar de tamanho conforme o tamanho e resolução do dispositivo. Já observou que os campos de entrada de texto do Android nunca se deformam, mesmo nos diferentes tipos de tela? Pois é! Eles usam 9 Patch para isso.

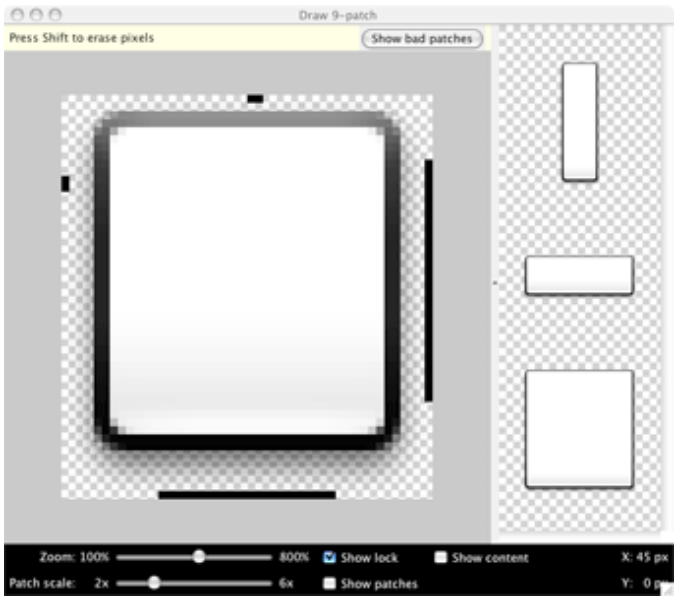


A imagem acima foi retirada da própria documentação do Android. O segredo dessa imagem são essas linhas nas extremidades da imagem. Com elas, você define a área que pode ou não ser esticada nessa imagem e o espaçamento.

As linhas da esquerda e do topo definem as áreas que serão esticadas. Na imagem acima, observe que estamos deixando as bordas intactas. Elas não podem ser esticadas.

As linha da direita e de baixo definem o espaçamento do conteúdo. Imagine que esta imagem será um botão. Observe a imagem da parte de baixo. O retângulo rosa no meio define a área que pode conter o texto do botão. Se o texto passar dessa área, significa que ele precisa esticar a imagem. Aí, ele estica obedecendo as linhas do topo e da esquerda. Entendeu? :-)

E como você faz essa imagem? Você cria elas no seu editor de imagens preferido, mas pode, e até sugerimos, que use uma ferramenta da própria SDK do Android, chamada Draw 9patch. Ela é interessante, pois já faz um preview para diversos casos.



A imagem acima é da ferramenta. No centro, temos a nossa imagem. As linhas nas extremidades nós criamos usando essa ferramenta. Na direita, temos os previews no caso de haver necessidade de esticar a imagem.

Essa ferramenta encontra-se no diretório do SDK do Android, mais especificamente no subdiretório **tool**.

Sobre o Autor

Marlon Silva Carvalho

É um programador de longa data. Já peregrinou por diversas linguagens e hoje se considera um programador agnóstico. Atualmente, está fascinado pelo mundo mobile e suas consequências para a humanidade. Está mais focado no desenvolvimento para Android, embora também goste de criar aplicativos para iOS.

Trabalha em projetos sobre mobilidade no SERPRO, tendo atuado no projeto Pessoa Física, para a Receita Federal do Brasil. Você pode encontrar ele no Twitter, no perfil @marlonscarvalho, em seu blog, no endereço http://marlon.silvacarvalho.net e através do e-mail marlon.carvalho@gmail.com.

Sobre os Cards

Caso você já tenha visto os excelentes RefCards da DZone, deve ter percebido a semelhança. E é proposital. A ideia surgiu após acompanhar estes RefCards por um bom tempo. Contudo, eles são mais gerais. O objetivo destes Cards é tratar assuntos mais específicos. Trazendo informações relevantes sobre eles.

Caso tenha gostado, continue nos acompanhando! Tentaremos trazer sempre novos cards sobre os mais variados assuntos. Inicialmente, focaremos no Android.

O ícone principal deste card é de autoria de Wallec e foi obtido em seu profile no DeviantArt.
http://wwalczyszyn.deviantart.com/