



Android - Cards de Referência

Recursos

Bem vindo ao nosso quarto card sobre Android! :) Desta vez, vamos conversar sobre os Recursos. Todo projeto possui recursos associado a eles, sejam imagens, layouts, cores entre outros. Vamos descobrir um pouco mais sobre estes recursos, suas peculiaridades e como usar eles.

SOBRE OS RECURSOS

Todo projeto de respeito é repleto de recursos. :) Sejam eles imagens, vídeos, músicas, etc. Muita coisa pode ser considerada como um recurso e o Android permite que você gerencie isto de forma fácil, podendo adaptar conforme mudanças de configuração do dispositivo.

Conforme já conversamos em cards preliminares, você deve colocar seus recursos em subpastas da pasta **/res**. Estas subpastas podem receber sufixos conforme determinadas características que você deseja tratar: orientação de tela, tamanho de tela, dpi, localização e etc.

As possíveis subpastas que você encontrará em **/res** são:

animator e **anim** - coloque nestas pastas os arquivos XML que definem animações. A primeira é para as animações chamadas Property Animations, enquanto a segunda é para as animações chamadas Tween Animations.

color - coloque nestas pastas os arquivos XML que definem uma lista de cores a depender de determinados estados.

drawable - coloque nesta pasta os arquivos de imagens, que podem ser PNG, Nine Patch, JPG e GIF.

layout - esta pasta é específica para você guardar os arquivos que definem o layout de suas telas.

menu - aqui estão os arquivos em formato XML que definem como serão os menus que fazem parte do seu aplicativo.

values - esta pasta é multi-uso! Mas aqui você encontrará um arquivo de muita importância, que é o **strings.xml**. Nele você colocará todos os textos que você quer exibir para o usuário. Você também encontrará o arquivo **dimen.xml**, **styles.xml** e **colors.xml** que definem dimensões, estilos e cores estáticas.



DICA

Os arquivos de recurso devem ser sempre em minúsculas e não podem conter a maioria dos caracteres especiais. Apenas o _ (sublinhado).

CORES

Crie um arquivo chamado **colors.xml** dentro da pasta **/res/values** para que você possa definir constantes de cores. Essas constantes são definidas usando os formatos RGB e ARGB. Vamos ver um exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#ff0000</color>
    <color name="translucent_red">#80ff0000</color>
</resources>
```

Na primeira entrada, temos a cor vermelha, definida no formato RGB. No segundo caso, temos a cor vermelha também, mas com a definição do canal alpha, que define a transparência.

Só mais um detalhe, o nome do arquivo não importa muito, pode ser qualquer nome. Como boa prática, sugerimos que seja **colors.xml** dentro da pasta **/res/values/**.

CORES POR ESTADO

Imagine que você tem um botão na tela. Agora, você quer que a cor normal dele seja cinza. Ao usuário apertar ele, deve mudar a cor para preto. Como fazer isto? Usando a lista de cores por estado!

Crie um arquivo, com qualquer nome, dentro da pasta **/res/colors**. Agora, vamos usar as tags XML **<selector>** e **<item>**. Em **<item>**, nós temos alguns atributos que definem o estado e qual cor usar.

Um exemplo é melhor que mil palavras! Suponha que nós temos um arquivo **cores_botao.xml**.

```
<selector>

  <item android:state_pressed="true"
        android:color="#ff0000"/>

  <item android:state_focused="true"
        android:color="#0000ff"/>

  <item android:color="#000000"/>

</selector>
```

Definimos um `<selector>` e dentro temos várias entradas para `<item>`, uma para cada estado que queremos tratar. No primeiro `<item>`, estamos informando que ao botão ser pressionado, a cor deve ser branca. Os estados possíveis são: `state_pressed`, `state_focused`, `state_selected`, `state_checkable`, `state_checked`, `state_enabled`, `state_window_focused`.

No segundo caso, informamos que ao botão receber o foco, deve mudar a cor para azul. Por último, informa que a cor padrão deve ser preto. E agora? Como uso essa sequência de cores no botão? Veja o exemplo abaixo.

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/text"
    android:textColor="@color/cores_botao" />
```

REFERENCIANDO RECURSOS - 1

No exemplo anterior, você percebeu a forma como informamos qual será a cor do botão? Usamos **@color/cores_botao** no atributo **android:textColor**. Você vai perceber que esta é a forma que você tem de referenciar um recurso a partir de um layout ou, até mesmo, a partir de outros recursos.

Observe também o atributo **android:text**, que se refere a um texto no arquivo **strings.xml** usando **@string/text**. Você terá sempre um **@algumacoisa** para cada tipo de recurso.

TEXTOS

Agora, vamos adotar uma boa prática: remover todos os textos estáticos de nossos arquivos que definem layout ou lógica de apresentação. Vamos criar arquivos no formato XML que guardam esses valores.

Você pode definir três tipos de informações: strings simples, arrays de strings e plurais. Vamos entender elas.

Strings Simples

O nome do arquivo pode ser qualquer um, mas o mais comum é que o arquivo tenha o nome **strings.xml**. Vamos ver um exemplo deste arquivo.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

  <string name="chave">Valor!</string>

</resources>
```

Iniciamos o arquivo com a tag `<resources>`, seguida de uma sequência de tags `<string>`, definindo cada valor. E como referenciamos essas strings nos arquivos de layout? Usando o shortcut **@string/chave**.

Para obter o valor desta string em uma Activity, use o código a seguir.

```
String valor =
    getResources().getString(R.string.chave);
```

Array de Strings

E se você quiser exibir uma lista de dados pré-determinados? Você pode criar um Array de Strings que tem essa lista. Crie um arquivo, com um nome de sua escolha, na pasta **/res/values/**. Vamos criar um arquivo chamado **meu_array.xml**.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

  <string-array name="cidades">

    <item>Salvador</item>

    <item>São Paulo</item>

    <item>Rio de Janeiro</item>

    <item>Belo Horizonte</item>

  </string-array>

</resources>
```

Além disso usamos a tag `<string-array>` com um atributo `name`. Neste caso, estamos definindo uma lista com quatro cidades, através da tag `<item>`. E como usamos isso?

```
<Spinner
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/cidades"
    android:entries="@array/cidades"/>
```

O caso mais comum é carregar isto em um spinner, que é similar a um Combobox do Swing, ou Select de HTML.

Informamos no atributo **android:entries** o valor **@array/cidades**. Nós também podemos obter este array a partir de um código Java, veja o código a seguir.

```
String[] cidades =
    getResources().getStringArray(R.array.cidades);
```

Plurais

Este é um caso que você normalmente não se preocupa, mas que pode acontecer. Como você vai tratar a questão do plural em diferentes línguas? Por exemplo, se você faz um MP3 Player, você quer informar que tem “1 Música Seleccionada” ou “N Músicas Seleccionadas”. Em outras línguas isso muda também.

Muita gente prefere facilitar sua vida e escrever “2 Música(s) Seleccionada(s)” e deixar como parâmetro apenas a quantidade. Facilita, não é? Mas você é um cara chato e não acha bonito esses parênteses. :-)

Tem solução para o seu caso: use os plurais. Você define qual string será exibida a partir da quantidade que quer usar. Vamos ver um exemplo. Veja o arquivo **/res/values/strings.xml** abaixo.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <plurals name="quantidade_musicas">
        <item quantity="one">Uma música selecionada.</item>
        <item quantity="other">%d músicas selecionadas.</item>
    </plurals>
</resources>
```

Agora, vamos usar esse recurso em um código Java.

```
int count = getQuantidade();
Resources res = getResources();

String qtd = res.getQuantityString(R.plurals.quantidade_musicas,
    count, count);
```

Observe que não sabemos quanto será **count**, mas queremos uma string dependendo deste valor. Caso count seja igual a 1, então usará a primeira string, em todos os demais casos, usará a segunda string.

Mas você também pode ser mais específico ainda, pois o atributo quantity de <item> pode assumir os valores zero, one, two, few, many e other. Dê um tratamento específico para cada um destes, como você achar melhor.

REFERENCIANDO RECURSOS - 2

Opal! Usamos aí um tal de **R.java**, perceberam? Nos exemplos para obter strings, fizemos **R.array.cidades** e **R.string.valor**. Que arquivo é este? Ele é gerado automaticamente para você, não se preocupe. Trata-se de um arquivo de constantes, que relaciona todos os recursos que você coloca na pasta **/res**.

Ele é categorizado por tipo de recurso, neste caso, você terá constantes em R.strings, R.layout, R.drawable e por aí vai. Caso você abra este arquivo, que fica na pasta **/gen**, você verá uma estrutura parecida com a listagem a seguir.

```
public final class R {
    public static final class drawable {
        public static final int ic_launcher=0x7f020001;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
    public static final class style {
        public static final int AppTheme=0x7f050000;
    }
}
```

Observe que temos uma classe e constantes estáticas para cada recurso que temos. Você também encontrará nela o ID dos recursos. Por exemplo, caso coloque um Botão no seu layout e dê como **android:id** o valor **@+id/meu_id**, terá um arquivo R.java como o a seguir.

```
public final class R {
    public static final class id {
        public static final int meu_id=0x7f050000;
    }
}
```

Preste atenção quanto a essa questão do ID. Para ele aparecer no seu arquivo de recursos **R.java**, deve ser referenciado como **android:id="@+id/algumacoisa"**. Observe o sinal de +. Sem ele, não vai aparecer.

O mais importante é que você tenha sempre em mente que todo recurso contido na pasta **/res** estará referenciado neste arquivo para facilitar a referência a ele em seu código Java.

LAYOUT

Uma boa prática de programação é separar a lógica que define como a tela do aplicativo será daquela lógica que trata os eventos lançados pela interação do usuário com ela. Seguindo esta lógica, o Android permite que você crie arquivos de layout usando uma notação XML específica.

Estes arquivos devem constar na pasta **/res/layout** e podem ter o nome que você preferir, desde que siga as regras de nomenclatura.

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"/>

</RelativeLayout>
```

Temos acima a definição de uma tela que possui apenas um campo de exibição de texto. Lembra como associamos esse layout a uma Activity? Usando o método **setContentView(R.layout.activity_main)**, onde "activity_main" é o nome do arquivo que você criou para o layout.

MENU

Não vamos entrar em muitos detalhes com relação aos menus agora. Vamos apenas apresentar como são os arquivos XML que definem os menus, mas não vamos ver como usar eles na prática.

Os menus devem ser criados dentro da pasta **/res/menu** e os arquivos podem ter o nome que você desejar. Depois, eles podem ser referenciados em seu código usando **R.menu.entrada_menu**.

```
<menu>
    <item android:id="@+id/menu_settings"
        android:title="@string/menu_settings"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>
```

O código acima demonstra a definição de apenas um item de menu. As demais informações são discutidas no card específico sobre menus, que trata sobre os Menus de Contexto e Menus de Opções.

DIMENSÕES

Você também pode separar os valores de dimensões em um arquivo XML separado. Isto é útil e nós veremos o motivo de forma mais detalhada em breve. Lembre-se que todas pastas de recursos podem receber sufixos para que os recursos sejam usados de forma diferente a depender de configurações do dispositivo.

Isto inclui tamanho de tela, orientação e afins. Imagine que você quer referenciar uma altura e largura diferentes a depender da resolução! Você faz isso usando uma tag específica dentro de <resources>. A forma mais comum é criar um arquivo **dimens.xml** dentro da pasta **/res/values** e usar essas tags nele. Veja o exemplo abaixo.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <dimen name="textview_height">25dp</dimen>

    <dimen name="textview_width">150dp</dimen>

    <dimen name="ball_radius">30dp</dimen>

    <dimen name="font_size">16sp</dimen>

</resources>
```

Você referencie essas dimensões através das constantes em **R.dimen.*** ou em arquivos de layout com **@dimen/nome**.

BOOLEANOS

Você também pode definir valores booleanos em arquivos XML para serem acessados em outros arquivos de recurso ou no código Java. É comum que se crie um arquivo **/res/values/bools.xml** e coloque esses valores lá, conforme o trecho de código a seguir.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <bool name="screen_small">true</bool>

    <bool name="adjust_view_bounds">true</bool>

</resources>
```

Como referenciar no código Java? Usando **R.bool.screen_small**. Veja o código de exemplo logo abaixo.

```
Resources res = getResources();
boolean small = res.getBoolean(R.bool.screen_small);
```

INTEIROS

Além de booleanos, também podemos definir inteiros. A lógica é semelhante e normalmente criamos um arquivo **/res/values/integers.xml** para definirmos esses valores. Vamos a um exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="inteiro_1">1</integer>
    <integer name="inteiro_5">5</integer>
</resources>
```

ARRAY DE INTEIROS

Temos arrays de strings, então, nada mais coerente do que ter arrays de integer, certo? Claro. E temos. É exatamente da mesma forma que as strings, só muda uma tag. Vamos a um exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="umatedois">
        <item>1</item>
        <item>2</item>
    </integer-array>
</resources>
```

ARRAY DE RECURSOS

Array de Recursos? Complicou? É simples. E se você quisesse fazer um array em que os elementos apontam para outros recursos? É possível. Vamos a um exemplo para as coisas ficarem mais claras.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="icones">
        <item>@drawable/icone_1</item>
        <item>@drawable/icone_1</item>
        <item>@drawable/icone_3</item>
    </array>
</resources>
```

Neste exemplo temos um array que referência imagens. Contudo, você pode fazer uma “mistureba” de recursos. Poderia ter imagens misturadas com cores, dimensões e por aí vai. Só não é muito indicado, mas é possível.

E agora, como usar este array? No código Java, você usa a classe **TypedArray** para isso.

```
Resources res = getResources();
TypedArray icones = res.obtainTypedArray(R.array.icones);
Drawable drawable = icones.getDrawable(0);
```

ESTILOS

Use os estilos para definir um conjunto de propriedades visuais para elementos de tela. Por exemplo, você pode criar um estilo para todos os botões de sua aplicação. Os estilos são definidos em um arquivo XML separado e que normalmente é nomeado como **/res/values/styles.xml**. Vamos a um exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="Botao" parent="@android:style/Widget.Button">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#000000</item>
    </style>
</resources>
```

Observe que nas tags <item> colocamos no atributo name os nomes de atributos comuns em elementos de tela. Podemos também herdar as propriedades de um estilo-pai e modificar ou adicionar novos itens. Neste caso, estamos herdando o estilo padrão do botão e modificando o tamanho da fonte e a cor dessa fonte.

Para aplicar esse estilo em um botão, usamos a tag style. Veja o código abaixo.

```
<?xml version="1.0" encoding="utf-8"?>
<Button
    style="@style/Botao"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Salvar" />
```

DRAWABLES


Vamos ver agora os recursos de imagens. Basicamente, podemos usar imagens PNG, JPG, GIF e um tipo especial chamado 9-Patch. Veremos mais sobre esta última em um card específico.

As imagens ficam nas pastas **/res/drawable-***. Por que este *? Simples, normalmente dividimos as imagens.

Colocamos cada imagem conforme o tamanho de tela e DPI. Conversaremos mais sobre isto no próximo tópico. 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 que serão usadas a imagem de uma pasta específica. Você referencia essas imagens em seu código Java através das constantes contidas 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 bastante interessante.



DICA
Imagens são otimizadas durante o processo de build. As imagens nas pastas drawable-(ldpi|mdpi) podem ter sua compressão reduzida para economizar memória.

É tão interessante que achamos legal fazer um card especial sobre os recursos do tipo drawable! Vamos parar por aqui com relação aos drawables.

RECURSOS E DIFERENTES DISPOSITIVOS

Agora vem a parte interessante dos recursos. Você observou que tem as pastas drawable-mdpi, drawable-ldpi e por aí vai? Cada pasta desta pode ter um sufixo que especifica uma característica de configuração do dispositivo. O MDPI e o LDPI são referentes à densidade de pixels que o dispositivo possui.

Mas estas não são as únicas opções. Quanto à densidade de pixels, você pode ter ldpi, mdpi, hdpi e xhdpi. Mas também pode variar conforme o tamanho de tela, usando small, normal, large e xlarge. E com relação à versão do Android? Também pode, usando v14, v13 e por aí vai.

Interessante, não é? Mas fica mais interessante ainda quando você descobre que pode usar sufixos para definir Locales. Por exemplo, en e pt. E pode combinar tudo isso? Pode sim. Suponha que você quer fazer um layout de tela específico para tablets com densidade de pixels alta. Como ficaria? Assim: **/res/layout-xlarge-hdpi**.

E pode misturar tudo mesmo? Sim. O exemplo aí de cima ainda poderia ficar **/res/layout-en-xlarge-hdpi**. O que isto significa na prática? Que você pode criar recursos específicos para cada situação que quer tratar.

Mas também não fique assustado, você NÃO precisa criar recursos para exatamente cada tipo de dispositivo que existe.

Por exemplo, quando você cria imagens em drawable-hdpi, já está abrangendo um leque grande de dispositivos. Não se preocupe em ser extremamente detalhista. Quais sufixos qualificadores você pode usar? Vamos ver na tabela na página a seguir. Fique atento para um único detalhe: se você for misturar esses qualificadores, eles devem aparecer na ordem em que aparecem na tabela.

Configuração	Sufixo	Descrição
MCC e MNC	mcc310, mc310-mmcc004, mcc208-mnc00	O código móvel do país, opcionalmente acompanhado do código da rede. Caso o smartphone tenha uma conexão GSM, esta informação vem do cartão SIM.
Localização	en, fr, pt, etc	A localização configurada no dispositivo. Use para você alterar configurações para determinadas línguas.
Menor largura possível disponível	sw<N>dp	Menor largura possível disponível no dispositivo. Não altera em caso de mudança na orientação de tela. Use para garantir que mesmo que haja uma mudança na orientação, você quer ter garantido que terá esse largura mínima.
Largura Disponível	w<N>dp	Mínima largura disponível. Este valor altera conforme a orientação da tela.
Altura Disponível	h<N>dp	Mínima altura disponível. Este valor altera conforme a orientação da tela.
Tamanho da Tela	small, normal, large, xlarge	Tamanho físico da tela, baseado na medida diagonal.
Aspecto da Tela	long e notlong	Baseia-se no aspecto da tela. Se for uma tela larga, use long.
Orientação de Tela	port e land	Refere-se à orientação da tela, que pode ser landscape ou portrait.
Modo de UI	car, desk, television, appliance	Altera os recursos conforme o modo em que o dispositivo está sendo usado. Use “car” se o dispositivo estiver em uso em um dock de carro. Use “desk” se estiver em um dock de mesa. Já o “television” é para quando o dispositivo está conectada a uma TV.
Modo Noturno	night, notnight	Você pode alterar quais recursos quer exibir também baseado no modo de exibição noite ou dia.
Densidade	ldpi, mdpi, hdpi, xhdpi, nodpi, tvdpi	Baseado na densidade de pixels do dispositivo.
Tipo de Touch	notouch, finger	Use para alterar o visual conforme o tipo de interação do usuário com o smartphone. Caso seja com o toque na tela, use “finger”. Em caso de dispositivos sem tela sensível ao toque, use “notouch”.
Teclado	keysexposed, keyshidden, keyssoft	Baseia-se na presença ou não de um teclado físico ou virtual. Use “keysexposed” caso tenha um teclado virtual disponível, independente de estar visível, ou caso tenha um teclado físico e ele está visível. Quando você tem um teclado físico disponível mas ele fica escondido e não há um teclado virtual, use “keyshidden”. Use “keyssoft” no caso de haver um teclado virtual disponível, quer ele esteja visível ou não.
Entrada Primária	nokeys, qwerty, 12key	Baseia-se na forma como ocorre a entrada de texto. O “nokeys” é usado caso o dispositivo não tenha qualquer forma de fazer entrada de texto. O “qwerty” é para dispositivos que tem um teclado para a entrada de textos. O “12key” é para os dispositivos que tem um teclado alfanumérico para entrada de texto.
Navegação	navexposed, navhidden	Use esta opção para o caso de dispositivos que possuem uma tecla física de navegação na tela.
Navegação sem Touch	nonav, dpad, trackball, wheel	Forma padrão de navegação para dispositivos que não possuem tela sensível ao toque.
Versão Plataforma	v<N>	A versão da API do Android.



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/>

Este trabalho usa a licença **Creative Commons Attribution-NonCommercial-ShareAlike 3.0**

<http://creativecommons.org/licenses/by-nc-sa/3.0/>



Basicamente, você pode copiar e modificar, desde que redistribua suas modificações. Também não é permitido usar este material ou suas derivações de forma comercial.