

Conceitos de layout

Introdução

A primeira interação do usuário final da sua aplicação será através do layout, a parte gráfica que mostra o que o aplicativo tem a oferecer. É de conhecimento geral que se uma interface é má construída ou difícil de entender, é provável que a aplicação seja fechada e esquecida.

Interfaces gráficas bem implementadas, assim como a lógica da aplicação bem escrita, contribuem para uma melhor experiência de usuário e consequentemente para o uso contínuo do software.

Na plataforma Android, o layout de uma aplicação diz quais os elementos que uma interface possui e como se comportam. Todos os elementos no layout são construídos usando a hierarquia de **View** e **ViewGroup**.

View geralmente representa algo que o usuário pode ver e interagir na aplicação. Por exemplo, botões, áreas de texto, spinner, checkbox, radio buttons, etc. Já a **ViewGroup** é um tipo de elemento que contém views. **ViewGroup** diz como os elementos de interface se comportam na aplicação, porém não é visível. Apesar de presente na aplicação, o usuário final não consegue ver nem interagir com o **ViewGroup**.

Simplificadamente, onde os elementos são posicionados, qual o tamanho da largura e altura, qual o comportamento em resoluções específicas, cor, tipo e tamanho de fonte, animações e mais são construídos usando uma combinação de **View** e **ViewGroup**.

Objetos do tipo **View** são geralmente chamados de “widgets” e possuem comportamento diferente de acordo com o tipo. **ViewGroup** são geralmente chamados de “layouts” e definem a estrutura visual dos elementos. São exemplos de ViewGroup: **LinearLayout**, **RelativeLayout**, **ConstraintLayout**, **TableLayout**, **FrameLayout**, **ListView** e **GridView**.

Toda a combinação de **View** e **ViewGroup** constroem o layout de uma aplicação. Esses elementos são escritos em XML, uma linguagem de marcação que é consideravelmente simples de ler e compreender.

Layout Android

Abaixo, um exemplo de layout usando `LinearLayout` com um elemento de texto dizendo “Olá”.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text_size="14sp"
        android:text="Olá" />
</LinearLayout>
```

No exemplo acima, temos `LinearLayout` dizendo como o elemento `TextView` se comporta. No caso, os elementos filhos do `LinearLayout` se comportam verticalmente (`orientation="vertical"`).

Ainda no exemplo ao lado, `LinearLayout` é o `ViewGroup`, é ele que determina como os elementos que estão dentro dele se comportam.

Como regra, todo layout em Android possui exatamente um elemento raiz e deve obrigatoriamente ser do tipo `ViewGroup`. Isso significa que não é possível existir dois elementos `TextView` como elemento raiz ou ainda não é possível existir dois layouts `LinearLayout` como raiz.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
</LinearLayout>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
</LinearLayout>
```

Dois elementos `LinearLayout` como raiz, não é permitido.

Medidas

Além da declaração dos elementos de interface, é possível customizar tamanho de largura e altura, margens, tamanho de fonte, etc. As medidas na plataforma Android seguem uma boa prática de serem definidas em **sp** ou **dp**.

Scale-independent Pixels (sp)

Significa que a unidade é independente de pixels. Por qual motivo? Atualmente existem dispositivos de inúmeras resoluções, uns com muito mais pixels do que outro. Se a unidade de medida for definida em pixel, significa que aparelhos com mais pixels terão um tamanho de fonte menor e aparelhos com menos pixels terão um tamanho de fonte maior. A falta de padrão de interface não é uma boa prática a ser seguida.

A medida **sp** permite que o dispositivo Android desenhe o tamanho da fonte escalando o tamanho caso necessário. Ou seja, o dispositivo faz o cálculo baseado no tamanho da tela e determina a quantidade de pixels.

A medida **sp** é recomendada para fontes.

```
<TextView android:text_size="14sp" />
```

Density-independent Pixels (dp)

Segue exatamente o mesmo conceito que a unidade **sp**, no entanto a unidade **dp** é recomendada para tamanho de elementos.

```
<TextView android:layout_width="64dp" />
```

Além da medida **dp**, largura e altura também podem ser definidas de acordo com a hierarquia do elemento ou ao próprio elemento.

- **match_parent**

Quando um elemento possui este valor no atributo largura ou altura (width ou height), seu tamanho será igual o tamanho do elemento de nível acima, também comumente chamado de elemento pai.

- **wrap_content**

Quando um elemento possui este valor no atributo largura ou altura (width ou height), seu tamanho será o suficiente para acomodar seu conteúdo. Significa que o elemento cresce ou diminui de acordo com o conteúdo.

Dicas e boas práticas

Assim como qualquer sistema, mesmo que seja Java, Web ou C#, muitas informações na tela trazem um custo para o processamento. Android não é diferente, a quantidade de elementos na tela influencia o processamento e tempo de carregamento. Isso só falando de processamento, sem contar que uma tela com muitas informações pode não ser tão compreensível para o usuário.

Quanto mais elementos, maior o custo de processamento e memória.

Outro ponto importante é que mesmo que existam poucos elementos na tela, se eles estão dentro de muitos níveis de layout, isso também se torna custoso para o processamento.

Tentar sempre que possível, manter a hierarquia plana (flat).