

Padrão ViewHolder

Contextualização

O padrão ViewHolder surgiu junto com o elemento `ListView`, usado para fazer uma listagem de itens na aplicação. Este elemento era amplamente usado no desenvolvimento, mas tinha alguns problemas. O primeiro e talvez mais crítico era que uma listagem muito grande muitas vezes se tornava pesada e apresentava a sensação de que o aplicativo estava travando.

O motivo disso é que para cada linha da `ListView`, são buscados todos os elementos que essa linha contém através do método `findViewById()` que é exatamente o que tornava a performance ruim, pois esse método é custoso para a aplicação.

Para melhorar a performance, inicialmente o foco foi no método que era mais custoso, `findViewById()`. Sua finalidade é bem simples, buscar elementos da interface para serem manipulados com valores em texto, datas, imagens e assim por diante de acordo com cada item da listagem.

Contudo, notou-se que o objetivo da `ListView` poderia ser atingido mantendo somente uma referência aos elementos da listagem uma vez já buscados ao invés de buscar repetidas vezes o mesmo elemento.

Dessa maneira, surgiram alguns padrões de código que tentavam minimizar este custo e assim surgiu o conceito ViewHolder.

Considere o código abaixo:

```
EditText editReal = findViewById(R.id.edit_real);  
TextView textReal = findViewById(R.id.text_real);  
Button btnCalc = findViewById(R.id.button_calc);
```

Imagine a execução deste código diversas vezes durante uma listagem de 100 itens, por exemplo.

```
for (int i = 0; i < 100; i++) {  
    EditText editReal = findViewById(R.id.edit_real);  
    TextView textReal = findViewById(R.id.text_real);  
    Button btnCalc = findViewById(R.id.button_calc);  
  
    editReal.setText(R.string.teste);  
    textReal.setText(R.string.teste);  
    btnCalc.setText(R.string.teste);  
}
```

Muitas execuções e requisições para processar, correto? Nesse trecho percebemos a execução do `findViewById()` diversas vezes e poderia ocorrer mais vezes caso existissem mais itens.

Observe o trecho abaixo com uma abordagem diferente:

```
private ViewHolder mViewHolder = new ViewHolder();

private static class ViewHolder {
    private EditText editReal;
    private TextView textReal;
    private Button btnCalc;
}

private void bindData() {
    this.mViewHolder.editReal = findViewById(R.id.edit_real);
    this.mViewHolder.textReal = findViewById(R.id.text_real);
    this.mViewHolder.btnCalc = findViewById(R.id.button_calc);

    for (int i = 0; i < 100; i++) {
        this.mViewHolder.editReal.setText(R.string.teste);
        this.mViewHolder.textReal.setText(R.string.teste);
        this.mViewHolder.btnCalc.setText(R.string.teste);
    }
}
```

O que se tem agora é a criação de uma classe que armazena os elementos de interface para que não sejam buscados todas as vezes que forem manipulados. Existe então a reusabilidade de uma referência que torna o trecho mais performático, uma vez que não temos mais a execução do `findViewById()` repetidas vezes, mas somente uma vez.

Nota: A execução real da **ListView** é diferente do código apresentado, porém esse exemplo é usado para contextualizar o que acontece de maneira simplificada.

Uso durante o curso

Durante o curso são criadas várias activities e layouts. Cada um desses layouts possuem elementos de interface que são manipulados. Por que então não trazer o conceito de ViewHolder para a Activity?

Observe o código abaixo:

```
private static class ViewHolder {
    private EditText editReal;
}

private ViewHolder mViewHolder = new ViewHolder();

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.mViewHolder.editReal.setText(R.string.teste);
}
```

Fontes

1. Developer Android. **Create a List with RecyclerView**. Disponível em: <<https://developer.android.com/guide/topics/ui/layout/recyclerview>>. Acesso em 12 de junho de 2020.
2. Blog Alura. Utilizando o padrão ViewHolder. Disponível em <<http://blog.alura.com.br/utilizando-o-padrao-viewholder/>>. Acesso em 12 de junho de 2020.
3. Stackoverflow. **ViewHolder Pattern em uma Activity**. Disponível em <<https://pt.stackoverflow.com/questions/270541/viewholder-pattern-em-uma-atividade>>. Acesso em 12 de junho de 2020.
4. binPress. **Create a Fluid ListView with the ViewHolder Pattern**. Disponível em <<https://www.binpress.com/create-fluid-listview-viewholder-pattern/>>. Acesso em 12 de junho de 2020.
5. Zone. **Optimizing Your ListView with the ViewHolder Pattern**. Disponível em <<https://dzone.com/articles/optimizing-your-listview>>. Acesso em 12 de junho de 2020.