Android - Cards de Referência

Listas e Adapters

Bem vindo ao nosso décimo terceiro card sobre Android! :-) Agora, vamos conversar sobre as listas e adapters. Listas são praticamente indispensáveis em qualquer aplicativo Android.

LISTAS E ADAPTERS

Listas são recursos praticamente indispensáveis em qualquer aplicativo, seja qual for a plataforma. No Android, nós fazemos essas listas, muito conhecidas também como grids, usando um objeto do tipo **ListView** ou **GridView**.

Contudo, você precisa se lembrar que estamos falando de dispositivos com uma tela pequena, portanto, não espere encontrar funcionalidades como as que você encontra em aplicações Web e Desktop.

As ListViews são mais básicas e não possuem a ideia de colunas, apenas linhas. Tudo está contido em apenas uma coluna. Mas também temos o GridView que divide a Lista em um Array bidimensional.

E os **Adapters**? Eles lembram bastante, se é que não podemos dizer que são iguais, os datasources de outros ambientes. Eles fornecem os dados e como os dados serão exibidos dentro da lista.

Outro detalhe que lhe dá bastante liberdade na construção de listas: cada linha da lista é um layout em si. Ou seja, você pode "montar" como as linhas de sua lista serão. Você pode colocar uma imagem na esquerda, três linhas de texto no centro e um checkbox no final, por exemplo. Você define este layout também em um arquivo XML dentro da pasta **/res/layout**.

LISTVIEW E SIMPLEADAPTER

A forma mais simples de se exibir uma lista de dados na tela é usando a combinação de ListView com um **SimpleAdapter**. Vamos fazer um exemplo para tornar mais claro como usar eles.

O trecho de código a seguir demonstra a inclusão de um ListView no XML de Layout da nossa tela. Contudo, ele ainda não exibe informações, uma vez que precisamos associar a ele um Adapter.

Fazemos logo no segundo trecho de código, através de uma Activity que obtém uma referência para o elemento e usa um SimpleAdapter.

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

   <ListView
      android:id="@+id/listView"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
   </ListView>

</RelativeLayout>
```

O código logo abaixo pode parecer um pouco complicado, mas é bem simples. Vamos entender ele. Inicialmente, precisamos obter uma referência para o objeto ListView do arquivo de layout. Observe, logo no final do código, a criação de um SimpleAdapter. O construtor dele pede cinco parâmetros.

```
public class MainActivity extends Activity {
   private ListView listView;

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

      listView = (ListView) findViewById(R.id.listView);

      List<Map<String, String>> lista =
         new ArrayList<Map<String, String>>();

      Map<String, String> value1 =
         new HashMap<String, String>();
      value1.put("rowid", "1");
      value1.put("col1", "Teste 1");

      lista.add(value1);

      String[] from = new String[] { "col1" };
      int[] to = new int[] { android.R.id.text1 };

      SimpleAdapter adapter =
         new SimpleAdapter(this,
         lista, android.R.layout.simple_list_item_1, from, to);
         listView.setAdapter(adapter);
   }
}
```

Primeiro, um contexto, que pode ser a própria Activity na qual ele se encontra. O segundo é uma lista com os dados que irão preencher a linha. O terceiro é o arquivo de layout que cada linha terá. Depois, é um mapeamento entre "campo" e "elemento de tela".

A lista de dados é apenas um **Map** de Strings. Colocamos como chave o nome da coluna. Neste caso, temos duas colunas, uma **rowid** e outra **col1**. Precisamos adicionar vários Maps para a lista. Cada Map representa uma linha da ListView.

Preenchemos o terceiro parâmetro com um layout que o próprio Android já possui. Ou seja, podemos criar o nosso ou usar um já pronto. Usamos o layout que está em **android.R.layout.simple_list_item_1**. Em seguida, precisamos fazer um mapeamento, já que temos elementos de tela, como Labels e valores que devem ir para algum destes campos, certo?

O layout que usamos possui apenas um campo do tipo TextView para se exibir dados. Então, fazemos esse mapeamento informando qual coluna irá preencher qual campo de tela.

LISTVIEW E ARRAYADAPTER

Uma forma um pouco melhor de se fazer listas é usando a combinação ListView e **ArrayAdapter**. Ela é bem útil para aquelas listas com opções bem delimitadas.

```
public class ArrayAdapterActivity extends Activity {
   private ListView listView;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      listView = (ListView) findViewById(R.id.listView);

      String[] objects = new String[] { "Item 1", "Item 2", "Item 3" };

      ArrayAdapter<String> adapter =
         new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
            android.R.id.text1,
            objects);
      listView.setAdapter(adapter);
   }
}
```

Continuaremos tendo apenas uma ListView na tela, o que muda é apenas o código do Adapter. Desta vez, estamos usando um ArrayAdapter preenchido com apenas três Strings.

Inicialmente, criamos um Array de Strings com os itens que passaremos como parâmetro no construtor do ArrayAdapter.

O ArrayAdapter tem vários construtores disponíveis, neste caso, estamos usando um dos mais completo deles. Você também perceberá que existe outro construtor que, em vez de receber um Array de Strings, pode receber uma **List**.

O primeiro parâmetro é o contexto. O segundo é o layout de cada linha da ListView. O terceiro é o widget que conterá o valor da String. O quarto e último são os dados.

LISTVIEW E BASEADAPTER

Caso você queira ter mais controle na forma como será seu Adapter, então, você precisa ter sua própria classe estendendo de **BaseAdapter**. Esta é a forma mais flexível possível de se criar Adapters. Contudo, um pouco mais trabalhosa.

```
public class MeuAdapter extends BaseAdapter {
   private List<String> lista = new ArrayList<String>();
   private Context context;

   public MeuAdapter(Context context) {
      this.context = context;
      lista.add("Teste 1");
      lista.add("Teste 2");
   }

   public int getCount() {
      return lista.size();
   }

   public Object getItem(int position) {
      return lista.get(position);
   }

   public long getItemId(int position) {
      return position;
   }

   public View getView(int position, View convertView,
         ViewGroup parent) {

      if (convertView == null) {
         convertView =
            LayoutInflater.from(context).
               inflate(android.R.layout.simple_list_item_1, null);
      }

      TextView textView =
         (TextView) convertView.findViewById(android.R.id.text1);
      textView.setText(lista.get(position));

      return convertView;
   }
}
```

O trecho de código anterior demonstra os métodos que precisamos sobrescrever desta classe. São eles:

getCount(): retorne aqui a quantidade de linhas que sua ListView terá.

getItem(int position): retorne qual o item que contém os dados que preenchem esta linha.

getItemId(int position): retorne o ID do item que contém os dados que preenchem esta linha.

getView(): retorne a View que será usada para preencher esta linha.

O métod mais importante aqui é o **getView()**. Ele tem três parâmetros. A posição da linha que se quer exibir. A View daquela linha, que pode ser NULL caso seja a primeira vez a exibir ela. Por último, a View que é a pai.

Começamos checando se o parâmetro **convertView** é nulo. Se sim, é a primeira vez que a linha está sendo exibida. Neste caso, vamos fazer um **inflate** de um layout em XML. Aqui pode ser qualquer layout, lembre-se! Neste caso, estamos usando o **android.R.layout.simple_list_item_1**.

Precisamos de um **LayoutInflater** para fazer esse "inflate". Fazemos isso chamado o método estático **from()** desta mesma classe. Observe que ela pede um Contexto como parâmetro. Passamos o contexto que pegamos através do construtor da nossa classe de Adapter.

O método **inflate()** de **LayoutInflater** recebe como parâmetro o ID do recurso que contém o layout e a View raiz. Em seguida, com a View em mãos após o processo de **inflate**, vamos obter uma referência para o campo TextView e colocar nele um texto.

Ainda poderíamos ter métodos extras, específicos para o nosso caso. Um exemplo são métodos como **add()**, **addAll()**, **remove()** e etc.

LISTVIEW E SIMPLECURSORADAPTER

Também temos uma classe de Adapter específica para exibir dados de cursores. Elas são bastante úteis para exibir dados oriundos de uma tabela do SQLite. No exemplo que usaremos nesta seção, contudo, usaremos

um **MatrixCursor** para facilitar nossa vida. Entretanto, nada precisaria ser mudado no caso de se usar um cursor proveniente de um **SQLiteDatabase**. Vamos ao exemplo, que consta na próxima listagem de código.

```
public class SimpleCursorAdapterActivity extends Activity {
   private ListView listView;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      listView = (ListView) findViewById(R.id.listView);
      String[] columns = new String[] { "_id", "col1" };

      MatrixCursor cursor = new MatrixCursor(columns, 2);
      cursor.addRow(new String[] { "1", "Item 1" });
      cursor.addRow(new String[] { "2", "Item 2" });
      cursor.addRow(new String[] { "3", "Item 3" });

      String[] from = new String[] { "col1" };
      int[] to = new int[] { android.R.id.text1 };

      SimpleCursorAdapter adapter =
         new SimpleCursorAdapter(
            this,
            android.R.layout.simple_list_item_1,
            cursor,
            from,
            to,
            0);
      listView.setAdapter(adapter);
   }
}
```

Vamos analisar o que temos de diferente dos demais exemplos que vimos até este momento. Primeiro, precisamos de uma instância de MatrixCursor, que recebe como parâmetro no seu construtor um Array de Strings informando os nomes das colunas e a quantidade inicial de colunas deste Cursor (lembre-se um cursor possui dados como se fossem retornados de uma tabela de banco dados).

Colocamos duas colunas: _id e col1. Por que esse _id? Ele é necessário. Caso não tenha ele, o **SimpleCursorAdapter** lançará uma exceção. Ele sempre procura por essa coluna. Em seguida, usamos o método addRow() para adicionar dados ao Cursor.

O construtor do **SimpleCursorAdapter** não é muito diferente daquele que vimos anteriormente. O primeiro parâmetro é o contexto, o segundo o layout da linha, o terceiro o cursor. Depois dois parâmetros fazendo o mapeamento de coluna do cursor para objetos de tela.

GRIDVIEWS

Então, o que muda de ListView para GridView? Elas são idênticas em quase tudo. Usam adapters da mesma forma. A diferença é a forma como os dados serão organizados na lista.

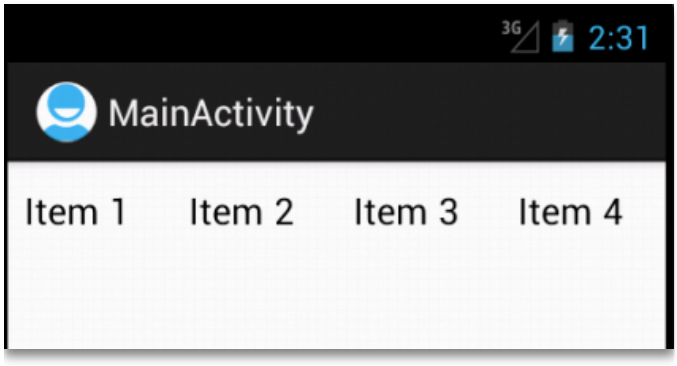
Na GridView, temos uma divisão em linhas e colunas. Você informa quantas colunas o seu Grid terá e cada célula será preenchida com o retorno do método **getView()** do Adapter (lembra do BaseAdapter?). Desta forma, você não tem como saber exatamente qual coluna está preenchendo a partir do Adapter.

Considere os Adapters que já criamos até agora. Vamos considerar o SimpleCursorAdapter da listagem abaixo. A única diferença que teremos é que usaremos um elemento <GridView> no arquivo de Layout.

```
public class SimpleCursorAdapterActivity extends Activity {
   private GridView gridView;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      gridView = (ListView) findViewById(R.id.listView);
      String[] columns = new String[] { "_id", "col1" };

      MatrixCursor cursor = new MatrixCursor(columns, 2);
      cursor.addRow(new String[] { "1", "Item 1" });
      cursor.addRow(new String[] { "2", "Item 2" });
      cursor.addRow(new String[] { "3", "Item 3" });
      cursor.addRow(new String[] { "4", "Item 4" });

      String[] from = new String[] { "col1" };
      int[] to = new int[] { android.R.id.text1 };

      SimpleCursorAdapter adapter =
         new SimpleCursorAdapter(
            this,
            android.R.layout.simple_list_item_1,
            cursor,
            from,
            to,
            0);
      gridView.setAdapter(adapter);
   }
}
```

Considerando o GridView abaixo, teremos como resultado uma única linha contendo quatro colunas, conforme a próxima imagem.

```
<GridView
   android:id="@+id/gridView"
   android:numColumns="4"
   android:gravity="center"
   android:columnWidth="50dp"
   android:stretchMode="columnWidth"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >
</GridView>
```



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/