Android 开发中使用 RecyclerView 实现聊天 APP 消息框左右对齐的效果

张寅森(千里冰封)

像 QQ、微信、短信等聊天 APP 的聊天界面中，都有一个特征，就是用户发送的消息呈现在右边，其他消息在左边。这个效果可以十分简单地通过 Android 中 v7 包里的 RecyclerView 实现。

RecyclerView 本身高度解耦，将大多数操作交给开发者自行完成，比起 ListView、GridView 等控件定制性更强。然而这也不可避免地增加了代码量，比如实现一个 RecyclerView 需要一下几步操作。虽然这里只有四行，但也只是最最核心的四行，真正的实现行数可能高达四位数呢。

```
messageRecycler = (RecyclerView) findViewById(R.id.messagesRecycler);
messageRecycler.setLayoutManager(new LinearLayoutManager(this));
messageRecycler.setItemAnimator(new DefaultItemAnimator());
messageRecycler.setAdapter(adapter);
```

第一行找到该 View，第二行设置一个布局管理器。这里我们编写一个聊天界面，所以用线性布局的管理器即可。第三行设置的是元素增删的动画，使用系统默认的即可。系统默认的动画较为简洁，并且十分符合 Material Design 风格，而且自行编写工作量也很大，系统自带的简洁动画代码 655 行，自行实现难度可见一斑。最后一行是设置适配器。这个 adapter 是我自行定义的适配器，源代码将在下文中提到。

先讲布局吧。由于 RecyclerView 的元素不能设置靠左靠右，于是我做了一些处理。这是原本的代码：

```
<android.support.v7.widget.CardView
    app:cardCornerRadius="@dimen/myMessageCornerRadius"
    app:cardElevation="3dp"
    android:id="@+id/messageCard"
    tools:layout_margin="@dimen/messageToSurround"
    tools:layout_gravity="end"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        tools:text="@string/hello"
        android:id="@+id/messageText"
        android:layout_margin="@dimen/myMessageMargin"
        android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</android.support.v7.widget.CardView>
```

非常简单，一个 CardView 一个 TextView。
效果如下。

但是在经过我的尝试之后，发现不能在代码中动态设置 layout_gravity 属性，况且 CardView 是元素的根 View，自然无法改动。所以我在经过了一阵头脑风暴之后，将代码写成了这样：

```
<FrameLayout
    android:id="@+id/messageFrame"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <android.support.v7.widget.CardView
        app:cardCornerRadius="@dimen/myMessageCornerRadius"
        app:cardElevation="3dp"
        android:id="@+id/messageCard"
        tools:layout_margin="@dimen/messageToSurround"
        tools:layout_gravity="end"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <TextView
            tools:text="@string/hello"
            android:id="@+id/messageText"
            android:layout_margin="@dimen/myMessageMargin"
            android:textSize="20sp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

    </android.support.v7.widget.CardView>
</FrameLayout>
```

只是在外面再套了一层 FrameLayout，但是这样的话，RecyclerView 的元素就不再是一个个的 CardView，而是一个个的 FrameLayout。而我在这一个个的 Frame 就能轻松自如地操作这一个个 CardView 了。

大致思想是这样：RecyclerView 的根元素是一个个横向 match_parent 的 FrameLayout，而这些 FrameLayout 至少作为根元素来说是一样的。而我需要操作的是里面的 CardView。我让 CardView 靠左靠右，都是 FrameLayout 里面的事，而之前的裸 CardView 就没有这么自由了，因为它是由 RecyclerView 动态生成的，内部原因细节请读者自行找寻，该内容不是本文讨论的重点。

说到这里就要引入 Java 代码了。

前面提到，一个 RecyclerView 需要一个 Adapter。这个 Adapter 是一个需要你自己实现的一个抽象类。它同时需要一个 ViewHolder，而 ViewHolder 同样也是一个抽象类，也需要开发者自行实现。

由于该 Adapter 和 ViewHolder 是针对单个 Activity 而言的，所以大可将这两个类作为 Activity 的子类。官方的实现中这两个抽象类也是 RecyclerView 的子类。

下面是我的代码。首先给出 ViewHolder。

```
class MessageViewHolder extends RecyclerView.ViewHolder {
    private CardView cardView;
    private TextView textView;
    public MessageViewHolder(View itemView) {
        super(itemView);
        cardView = (CardView)
                itemView.findViewById(R.id.messageCard);
        textView = (TextView)
                itemView.findViewById(R.id.messageText);
    }
    public void init(MyMessage message){
        FrameLayout.LayoutParams params =
                new FrameLayout.LayoutParams(
                        RelativeLayout.LayoutParams.WRAP_CONTENT,
                        RelativeLayout.LayoutParams.WRAP_CONTENT
                );
        if(message.isFromSaber()){
            params.gravity = Gravity.START;
            // 左 上 右 下
            params.setMargins(15,15,30,15);

            cardView.setCardBackgroundColor(
                    getResources().
                            getColor(R.color.cardColor1));
            textView.setTextColor(
                    getResources().
                            getColor(R.color.cardColor1Pressed));
        }
        else {
            params.gravity = Gravity.END;
            // 左 上 右 下
            params.setMargins(30,15,15,15);

            cardView.setCardBackgroundColor(
                    getResources().
                            getColor(R.color.cardColor5));
            textView.setTextColor(
                    getResources().
                            getColor(R.color.cardColor5Pressed));
        }
        cardView.setLayoutParams(params);
        textView.setText(message.getMessage());
    }
}
```

Init 方法是我自己搞出来给自己用的。作用其实就是将复杂的对话框颜色设置的代码封装起来，在外部能更好、更简洁地调用。这里 init 的作用是根据消息的发送对象决定消息框的颜色以及方向。在这里就封装了解决之前的问题的代码。

然后初始化一个 Adapter。

```
class MessageAdapter extends RecyclerView.Adapter {
    private OnItemClickListener onItemClickListener;
    public void setOnItemClickListener(
            OnItemClickListener onItemClickListener){
        this.onItemClickListener = onItemClickListener;
    }
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        return new MessageViewHolder(LayoutInflater.from(MainActivity.this)
                .inflate(R.layout.message, parent, false));
    }
    @Override
    public void onBindViewHolder(final RecyclerView.ViewHolder holder, int position)
{
        ((MessageViewHolder)holder).init(
                brain.getMessageByPosition(position)
        );
        if (onItemClickListener != null) {
            holder.itemView.setOnClickListener(
                    new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    onItemClickListener.onItemClick(holder.itemView,
                            holder.getLayoutPosition());
                }
            });
            holder.itemView.setOnLongClickListener(
                    new View.OnLongClickListener() {
                @Override
                public boolean onLongClick(View v) {
                    onItemClickListener.onItemLongClick(holder.itemView,
                            holder.getLayoutPosition());
                    return false;
                }
            });
            holder.itemView.setOnTouchListener(
                    new View.OnTouchListener() {
                @Override
                public boolean onTouch(View v, MotionEvent event) {
                    onItemClickListener.onItemTouch(holder.itemView,
```

```
                                holder.getLayoutPosition(), event);
                        return false;
                    }
                });
            }
        }
        @Override
        public int getItemCount() {
            return brain.getDataSize();
        }
}
```

这个 Adapter 里面放着一个 ArrayList，变量名 data，用于存储数据。一个监听器，使用接口实现控制反转。在 `public void onBindViewHolder(final RecyclerView.ViewHolder holder, int position)` 方法中，直接调用 ViewHolder 的 init 方法，传入发送者的参数即可。非常简洁。本文未提到的方法，均为按照常规编写的方法，不在本文赘述。

附带一个字段封装的类 MyMessage，由于早期 APP 的虚拟角色是 Saber，所以在变量名中也有体现。现在的形象是暗杀教室中的小律。

```
package util;
/**
 *  数据字段的封装。
 * Created by Administrator on 2016/1/6 0006.
 */
public class MyMessage {
    public static final String ID = "id";
    public static final String MSG = "msg";
    public static final String FROM_SABER = "fromSaber";
    public static final int IS_FROM_SABER = 1;
    private boolean fromSaber;
    private boolean idAvailable;
    private String message;
    private int id;
    public MyMessage(boolean fromSaber, String message) {
        this.fromSaber = fromSaber;
        this.message = message;
        idAvailable = false;
    }
    public MyMessage(int fromSaber, String message) {
        this.fromSaber = fromSaber == IS_FROM_SABER;
        this.message = message;
        idAvailable = false;
    }
    public MyMessage(int fromSaber, String message, int id) {
        this.fromSaber = fromSaber == IS_FROM_SABER;
        this.message = message;
```

```
            idAvailable = true;
            this.id = id;
        }
        public MyMessage(boolean fromSaber, String message, int id) {
            this.fromSaber = fromSaber;
            this.message = message;
            idAvailable = true;
            this.id = id;
        }
        public int getId() {
            return idAvailable ? id : 0;
        }
        public boolean isFromSaber() {
            return fromSaber;
        }
        public String getMessage() {
            return message;
        }
        public boolean isIdAvailable() {
            return idAvailable;
        }
}
```

其中一些变量如 idAvailable 涉及到数据库操作的层面，不属于本文讨论范畴。在此略过。

这样就简单实现了一个聊天 APP 中消息左右对齐的界面效果。

Material Design 的 UI 一直是我的最爱，无论多累都应该实现，唯美的卡片式 UI！

总结：本文提出一个理念，即通过 FrameLayout 嵌套 CardView 的方法实现左右靠拢的效果。由于大多数情况下原生控件无法满足需求，大多数开发者的第一方案是自定义 View，然后徒手写 3000+行代码，写得自己都看不懂，而本文的方法则是完全靠原生 UI 控件的完美组合，原生控件 BUG 率小得多，而且谷歌的设计师再怎么也比一般的普通程序员的美感更强吧，所以还是尽量使用官方给的工具进行开发。这也是我选择 Android Studio 作为 IDE 而不是 eclipse 的根本原因。这种方法更加适合个人开发，如果是团队的话那也无所谓，毕竟团队的力量伟大，而我是单干，所以倾向于“使用轮子”而非“重新造轮子”。

有时，换个角度，换个思想解决问题，能想到更好，更优的 solution。最后祝读者代码永不报错，客户永不改需求。

2016 年 1 月 25 日