# 介绍

MaterialDesign是5.0推出的新特性，使用v7包可以向下兼容，但是很多的MaterialDesign效果只在5.0的手机生效。对于MaterialDesign的兼容可以查看：<https://developer.android.com/training/material/compatibility.html>

MeterialDesign的使用：<https://developer.android.com/training/material/index.html> 主要介绍主题、RecycleView 、CardView。

Material Design是谷歌新的设计语言，谷歌希望寄由此来统一各种平台上的用户体验，Material Design的特点是干净的排版和简单的布局，以此来突出内容。

Material Design对排版、材质、配色、光效、间距、文字大小、交互方式、动画轨迹都做出了建议，以帮助设计者设计出符合Material Design风格的应用。

Material Design设计语言鼓励大家使用充满活力的鲜艳色彩，并在同一界面建议使用三种色调，并保障有一个强色调，强色一般处于处于视图最底层，例如状态栏或者actionbar。通过强色调形成鲜明的对比，更容易突出内容的重要性。对于文字色彩的取值，Material Design建议在浅色背景上采用黑色，在深色背景上采用白色。重要信息和标题采用87%透明度，次要文字采用54%透明度，而更次要的说明文字可以采用26%的透明度。对于想特别突出或者可点击的文字，建议使用强色调。不同层级的视图，可以通过阴影来凸显。对于带有操作且内容突出的区域，可以使用cardview进行隔离，对于内容不太重要或者操作比较单一的区域，可以使用分割线进行隔离。

# 主题

切换主题，其实就是在xml中定义多个主题，然后在Activity中调用setTheme方法切换主题即可。需要注意的是这个方法必须写在

一整套的主题颜色如何配色呢？可以使用这个网站生成：<http://www.materialpalette.com/red/deep-purple>

主题不仅可以对Application和Activity使用，也可以对某一个控件单使用，或者是在xml布局中给一个根节点控件设置android:theme属性，来修改它和它所有子控件的主题。 如果我们要对特定控件实例做自定义修改，更建议大家可以通过控件自身的API进行设置修改。

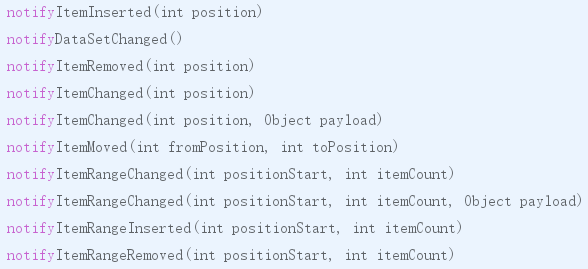
Material Design主题只有在API级别为21以上才可使用，在v7支持库中提供了部分控件的Material Design主题样式，如果想使应用在android的所有版本上都能统一风格，我们可以对控件效果做自定义或者使用一些第三方的兼容包。目前最有效的做法是针对21版本创建value－21资源目录，使用Material Design风格主题，在其他版本使用v7的Theme.AppCompat.Light风格主题。

# RecyclerView

* RecyclerViw、LayoutManager、Adapter、DataSet关系图：  
  
* LayoutManager有3个子类，在它们的构造方法中都可以指定布局的方向，其中表格和瀑布的布局管理器还可以指定列数。
* RecyclerView默认不显示滚动条，如要显示加上：android:scrollbars="vertical"
* RecyclerView没有setOnItemClickListener方法，需要点击事件可以在onBindViewHolder中设置单击事件。
* RecyclerView中item根元素中的layout\_打头的属性也是生效的。
* 在代码中如果要为根View设置大小，可使用：RecyclerView.LayoutParams
* 当删除删除adapter中的一条数据时，调用notifyItemRemoved方法可以显示动画，但随后得跟着调用notifyDataSetChanged，为了让上面的动画先显示，后面这个方法可以延迟点再调用。
* 如果你知道内容变化不会导致RecyclerView的大小改变，则使用此设置以提高性能：recyclerView.setHasFixedSize(true);
* 有一个SimpleStringAdapter，构造方法接收Context和String[]，会从cotenxt中取出主题中的selectableItemBackground所使用的颜色用于itemView，会把String[]转换为List，以便可以增加和删除。getValueAt(int position)返回指定位置数据。有swap(pos1, pos2)方法可交换数据中的位置，并调用notifyItemRemoved(pos1)和notifyItemInserted(pos2)进行通知。ItemView左右20dp的padding，并且上下左右都有margin值，每个值都不一样，显示item时会在前面加上position，每条item的最小高为：200 + 字符数 \* 10，item的颜色分别为黑、红、深灰、蓝。
* RecyclerView.ViewHolder中的itemView即我们的根itemView。
* RecyclerView的复用和ListView的复用不太相同，ListView创建的itemView数量为界面可见item + 1，而RecyclerView的itemView数量要比界面可见的多好几条。
* 要记清一条概念，有多少个itemView就会有多少个ViewHolder对象，它们是绑定关系，所以ViewHolder中的getAdapterPosition()方法可以得到它绑定的itemView的位置，当itemView复用的时候，系统会自动把ViewHolder中的poistion位置进行自动更新。比如原来是position为5的ViewHolder，现在复用到position 25的位置，则调用mViewHolder.getAdapterPosition() = 25。ViewHolder还可以获取到它所在postion的itemId、itemViewType。
* RecyclerView.Adapter中没有getItem方法，所以如果想要在外部获取Adapter中的数据，只能自己提供get方法。如果要设置点击监听器，可以在onCreateViewHolder中设置，如下：  
    
  假如会创建15个itemView，再多的数据也是复用这15个itemView，则它们会分别有对应的ViewHolder和OnClickListener，也就是说有15个itemView、15个ViewHolder、15个OnClickListener，而我们以前的ListView的单击事件需要一个，如果也想实现这个效果可以在onBindViewHolder中进行设置，如下：

**public** ViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {  
 **final** MyAdapter.ViewHolder viewHolder = **new** ViewHolder(View.*inflate*(parent.getContext(), android.R.layout.***simple\_list\_item\_1***, **null**));  
 viewHolder.**itemView**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **int** position = viewHolder.getAdapterPosition();  
 Toast.*makeText*(RecycleViewActivity.**this**, **"position = "** + position, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 **return** viewHolder;  
}

**public void** onBindViewHolder(ViewHolder holder, **final int** position) {  
 holder.**textView**.setText(**datas**.get(position));  
 holder.**itemView**.setTag(holder);  
 holder.**itemView**.setOnClickListener(**onClickListener**);  
}  
  
**private** View.OnClickListener **onClickListener** = **new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 ViewHolder viewHolder = (ViewHolder) v.getTag();  
 Toast.*makeText*(RecycleViewActivity.**this**, **"haha position = "** + viewHolder.getAdapterPosition(), Toast.***LENGTH\_SHORT***).show();  
 }  
};

* RecyclerView.Adapter中的notifyItemRemoved(position)方法只是通知某个位置被移除了，所以不会导致onCreateViewHolder和onBindViewHolder方法的调用，除非正好某个View要复用了，但这也只会导致onBindViewHolder方法的调用而已，而且复用一个就只调用一次，而notifyDataSetChanged()会导致所有的item都废弃，导致onCreateViewHolder和onBindViewHolder都重新执行，而且是一下执行很多次，效率低。同样的，nofityItemInserted(position)方法只会导致onBindViewHolder方法调用一次，而且onBindViewHolder中的position参数就是nofityItemInserted(position)中的position。下面还有很多的通知方法，没有研究，但是应该也是一样的原理，只对指定的位置的item进行增删改操作：  
  
* RecylerView具体Demo 看下：

**public class** RecycleViewActivity **extends** AppCompatActivity {  
 **private** String[] **datas** = {**"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"g"**, **"h"**, **"i"**, **"j"**, **"k"**, **"l"**, **"m"**, **"n"**, **"o"**, **"p"**, **"q"**};  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_recycle\_view***);  
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.***recycleView***);  
 recyclerView.setHasFixedSize(**true**); *// 如果你知道内容变化不会改变RecyclerView的大小，则使用此设置以提高性能* RecyclerView.LayoutManager layoutManager = **new** LinearLayoutManager(**this**);  
 recyclerView.setLayoutManager(layoutManager);  
 RecyclerView.Adapter adapter = **new** MyAdapter();  
 recyclerView.setAdapter(adapter);  
 }  
  
 **class** MyAdapter **extends** RecyclerView.Adapter<MyAdapter.ViewHolder> {  
 @Override  
 **public** ViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {  
 MyAdapter.ViewHolder viewHolder = **new** ViewHolder(View.*inflate*(parent.getContext(), android.R.layout.***simple\_list\_item\_1***, **null**));  
 **return** viewHolder;  
 }  
  
 @Override  
 **public void** onBindViewHolder(ViewHolder holder, **int** position) {  
 holder.**textView**.setText(**datas**[position]);  
 }  
  
 @Override  
 **public int** getItemCount() {  
 **return datas**.**length**;  
 }  
  
 **class** ViewHolder **extends** RecyclerView.ViewHolder {  
 TextView **textView**;  
 **public** ViewHolder(View itemView) {  
 **super**(itemView);  
 **textView** = (TextView) itemView;  
 }  
 }  
 }  
}

# CardView

CardView可拥有阴影和圆角。

这个控件的使用非常简单，想要把一个控件显示在卡片上，在外层包裹CardView即可。CardView属性：

* + app:cardBackgroundColor="#6c6cec" 卡片背景
  + app:cardElevation="24dp" 阴影海拔高度
  + app:cardCornerRadius="16dp" 圆解角度
  + app:contentPadding，不知道这个属性跟padding有什么区别

# View阴影

View阴影只在5.0及以上系统有效果。

阴影的外观由View的Z属性决定。z = elevation + translationZ，eveation为View的海拔高度，这是静止高度，我们可以在运行时给这个View执行移动动画，在z轴上移动。

elevation属性在xml中：android:elevation，在代码中：View.setElevation()

在z轴上移动：View.setTranslationZ()

ViewPropertyAnimator.z() 与 ViewPropertyAnimator.translationZ() 方法让您能够轻松地为视图高度添加动画。

您也可使用 StateListAnimator 以说明性方式指定这些动画。 当状态改变会触发动画（例如当用户按下按钮）时，此方法特别有用

在5.0之前，我们如果想给view添加阴影效果，以体现其层次感，通常的做法是给view设置一个带阴影的背景图片，现在，我们只需要简单的修改view的Z属性，就能让其具备阴影的层次感。

Z属性会扩大view的显示区域，如果它的大小大于或等于父视图的大小，那么它的阴影效果就无法显示了，view并不会因为z属性而把自身缩小腾出空间显示阴影。

# View的轮廓

在Android的世界里，所有的View都是矩形的，虽然可以给View设置背景圆形的图片，即可在界面显示出圆形的内容，但是View的大小实际上仍然是矩形，并且设置的图片也是实际上也是矩形，只是圆形意外的区域为透明色。

如果系统根据View的大小来为我们生成对应的阴影，有时候就会出现很奇怪的效果。

为了解决该类问题，View增加了一个新的描述来指明内容显示的形状，这就是轮廓。通过shape设置的背景，View会自动根据shape的形状进行轮廓判定，通过color设置的背景，View默认其轮廓和View的大小一样。但是通过图片进行背景设置，View则无法获知轮廓的形状，这个时候就需要我们程序员显示的指定。

在xml布局中，可以通过android:outlineProvider来指定轮廓的判定方式：

none 即使设置了Z属性，也不会显示阴影

background 会按照背景来设置阴影形状

bounds 会按照View的大小来描绘阴影

paddedBounds 和bounds类似，不过阴影会稍微向右偏移一点

# SwipeRefreshLayout

这个使用超级简单，使用这个布局把需要下拉刷新的控件包裹起来（可以包裹任意的控件，如LinearLayout）。在代码中调用这个控件的setOnRefreshListener方法传入监听器，用于监听是否可以去刷新数据，当数据刷新之后调用该类的setRefreshing(false)，让刷新的圈圈隐藏即可。

setColorSchemeColors(Color.RED,Color.GREEN,Color.BLUE);//设置进度样式，这里的参数是可变参数，可以设置任意个颜色，按顺序显示

setProgressBackgroundColorSchemeColor(Color.BLACK);//设置背景色

# ActionBar上可以设置一些功能

<**menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 tools:context=".MainActivity"**>  
 <**item  
 android:id="@+id/action\_search"  
 android:orderInCategory="100"  
 android:icon="@mipmap/ab\_search"  
 android:title="搜索"  
 app:actionViewClass="android.support.v7.widget.SearchView"  
 app:showAsAction="ifRoom"** />  
  
 <**item  
 android:id="@+id/action\_share"  
 android:orderInCategory="100"  
 android:icon="@mipmap/ab\_share"  
 android:title="分享"  
 app:actionProviderClass="android.support.v7.widget.ShareActionProvider"  
 app:showAsAction="ifRoom"** />  
</**menu**>

# 七.与旧版本兼容

1.可以通过Support Library使用的：

RecyclerView

SwipeRefreshLayout

CardView

Palette颜色识别

定义theme:让应用的theme继承Theme.AppCompat,

同时定义属性的时候，去掉android：，比如：



2.定义多个layout,使用新API的布局放在res/layout-v21中，其他的放res/layout

3.在代码中对系统Version做判断，使用对应的效果，如下：  
 