Android 沉浸式 UI 实现及原理

2016-07-05 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：Jude95
> 链接：http://www.jianshu.com/p/f3683e27fd94

沉浸式体验

首先别吐槽沉浸式这个名词吧,毕竟这各名字是广为人知并且比透明状态栏加透明导航栏更酷。充分使用整个屏幕将这2个系统视图融入自己APP也算沉浸式体验吧。

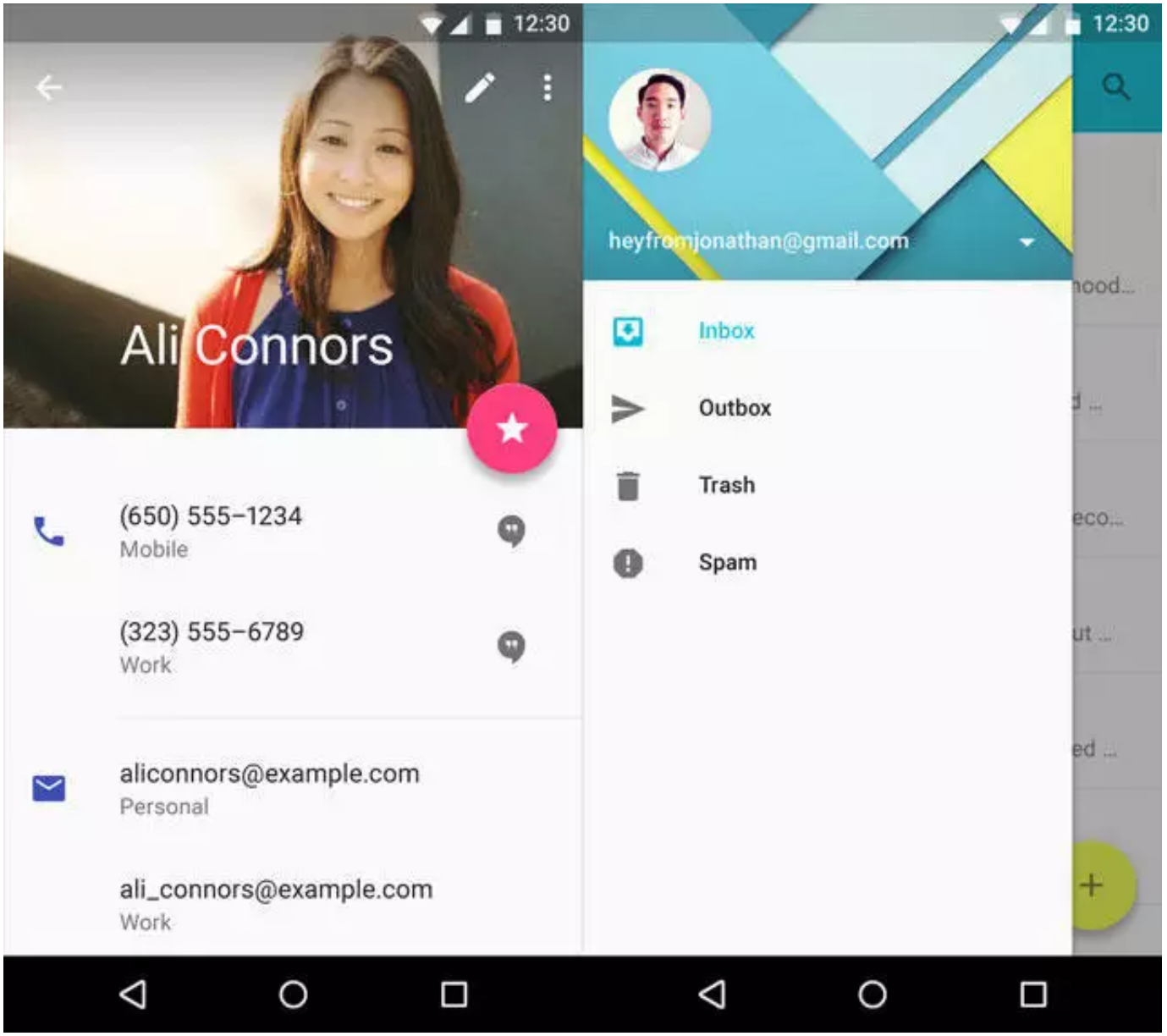
首先2个名词：
StatusBar：

Paste_Image.png

NavigationBar：

Paste_Image.png

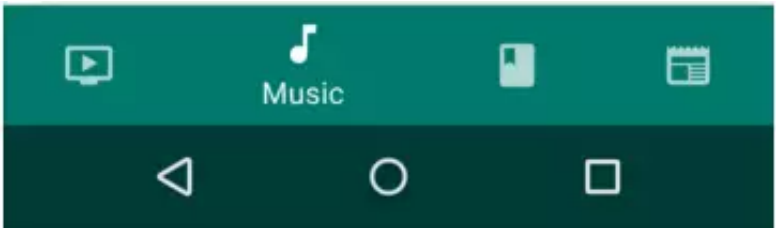
下面是Google的官方标准模版：



未标题-1.jpg

在官方示例中：
StatusBar是一个半透明阴影，View可以伸展到其后面。
NavigationBar是纯黑不能使用的。

Google提供NavigationBar的透明与使用的可能，却没有推荐使用。个人觉得是为了给

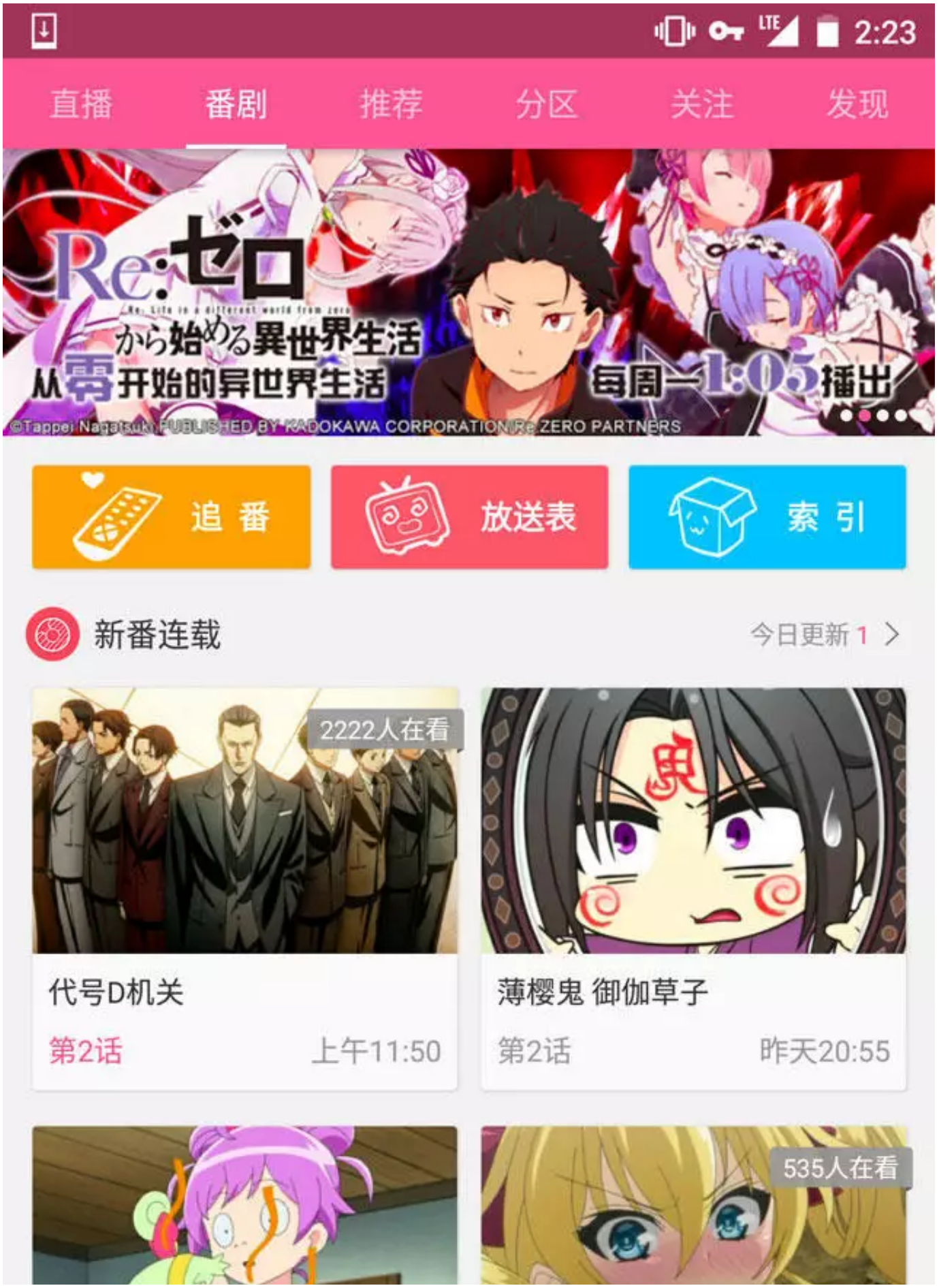


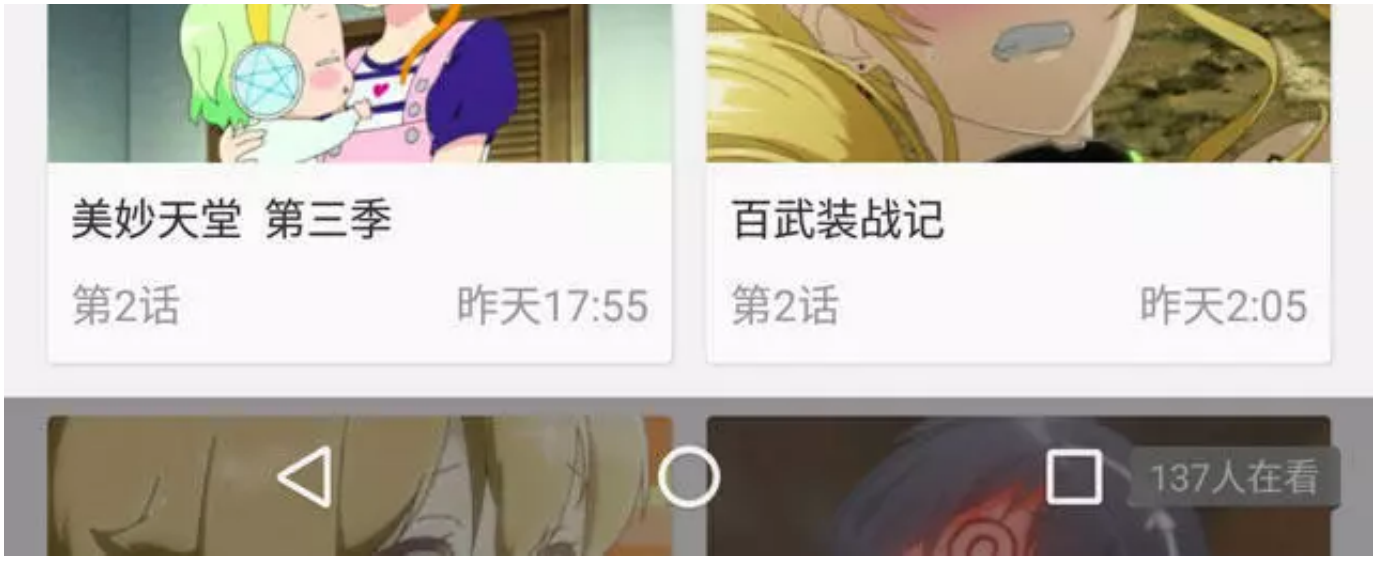
Paste_Image.png

Bottom navigation

做准备。这里不讨论Bottom navigation的优劣(我是Bottom navigation黑)。

下面是B站的：





QQ图片20160525195336.jpg

不可否认使用NavigationBar空间对多下巴手机是种拯救。
（Google自己推的虚拟按键，却自己放弃了屏占比这一大优势。）

好了，B站的UI就是我所期望的UI。下面我们就来实现它。

style的配置

android从4.4开始，开始支持UI使用StatusBar与NavigationBar的范围。
所以要进行下面的配置：

在value中的styles.xml中设置

```
<!-- Base application theme. -->
<style name="AppTheme.Base" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Customize your theme here. -->
</style>
<style name="AppTheme" parent="AppTheme.Base"></style>
```

在value-v19中的styles.xml中设置(为了兼容4.4)

```
<style name="AppTheme" parent="AppTheme.Base">
    <item name="android:windowTranslucentStatus">true</item>
    <item name="android:windowTranslucentNavigation">true</item>
</style>
```

在value-v21中的styles.xml中设置

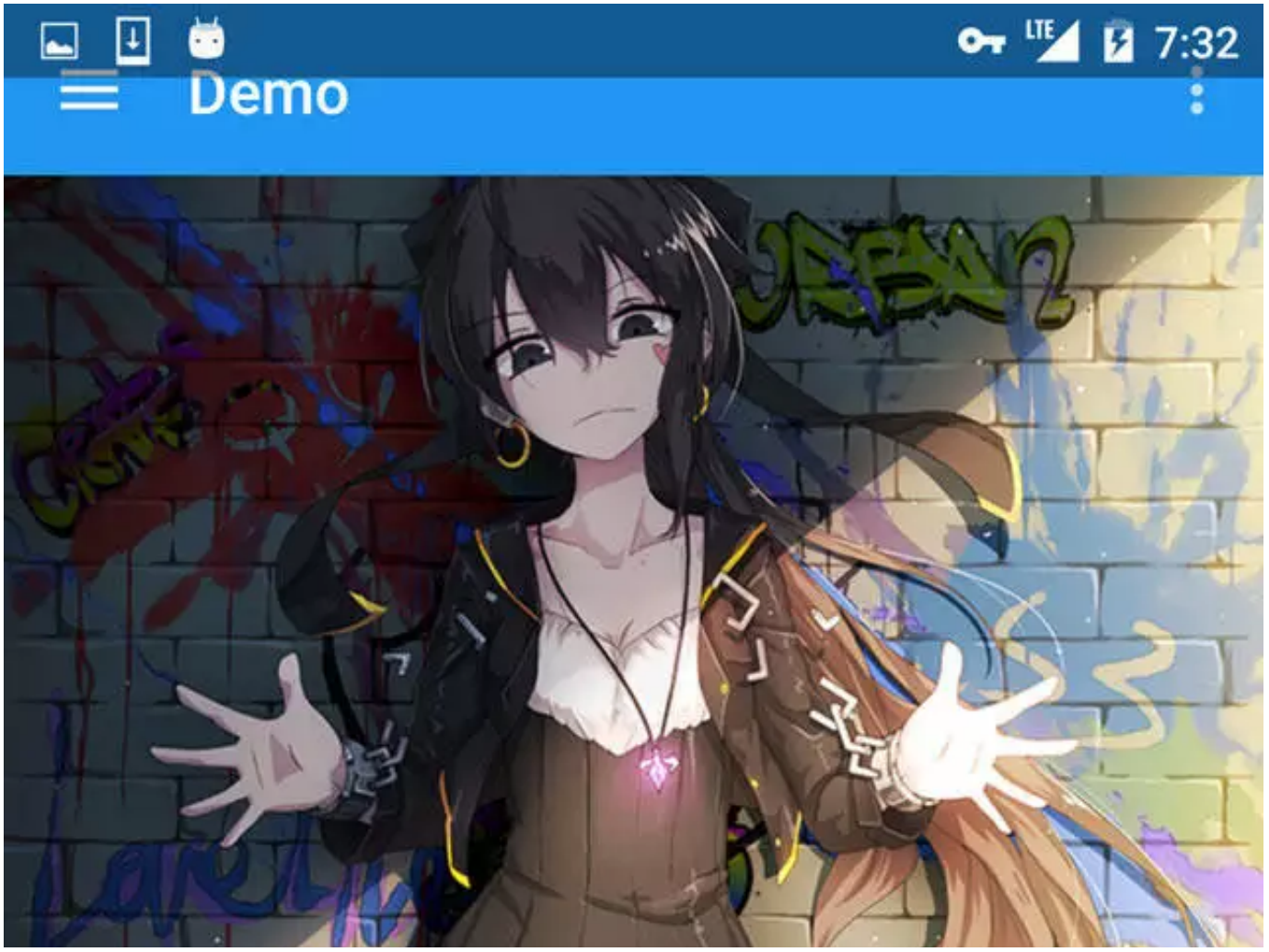
```
<style name="AppTheme" parent="AppTheme.Base">
    <!--透明状态栏-->
    <item name="android:windowTranslucentStatus">true</item>
    <!--透明导航栏-->
    <item name="android:windowTranslucentNavigation">true</item>
    <!--使状态栏，导航栏可绘制-->
    <item name="android:windowDrawsSystemBarBackgrounds">true</item>
</style>
```

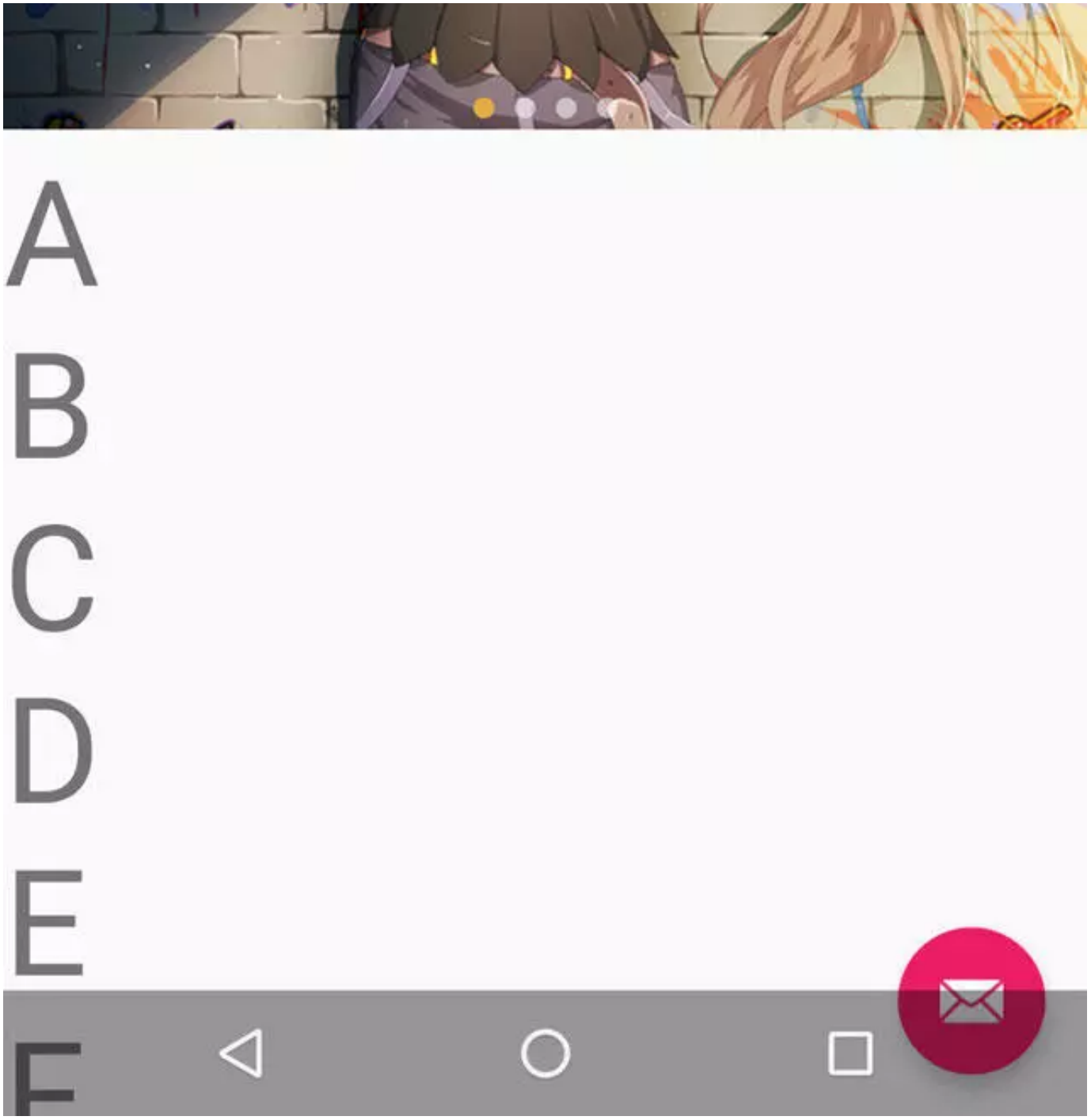
然后使用AppTheme这个主题，这是1个示例，应该看得出来吧。只要在你的AppTheme的v19版本和v21版本添加了相应属性就好。

使用ToolBar

当你使用了StatusBar的部分，就不要再使用ActionBar了。
现在基本都会使用ToolBar了吧。还不会可以参考 ToolBar的使用

然后效果如下：





QQ图片20160525195351.jpg

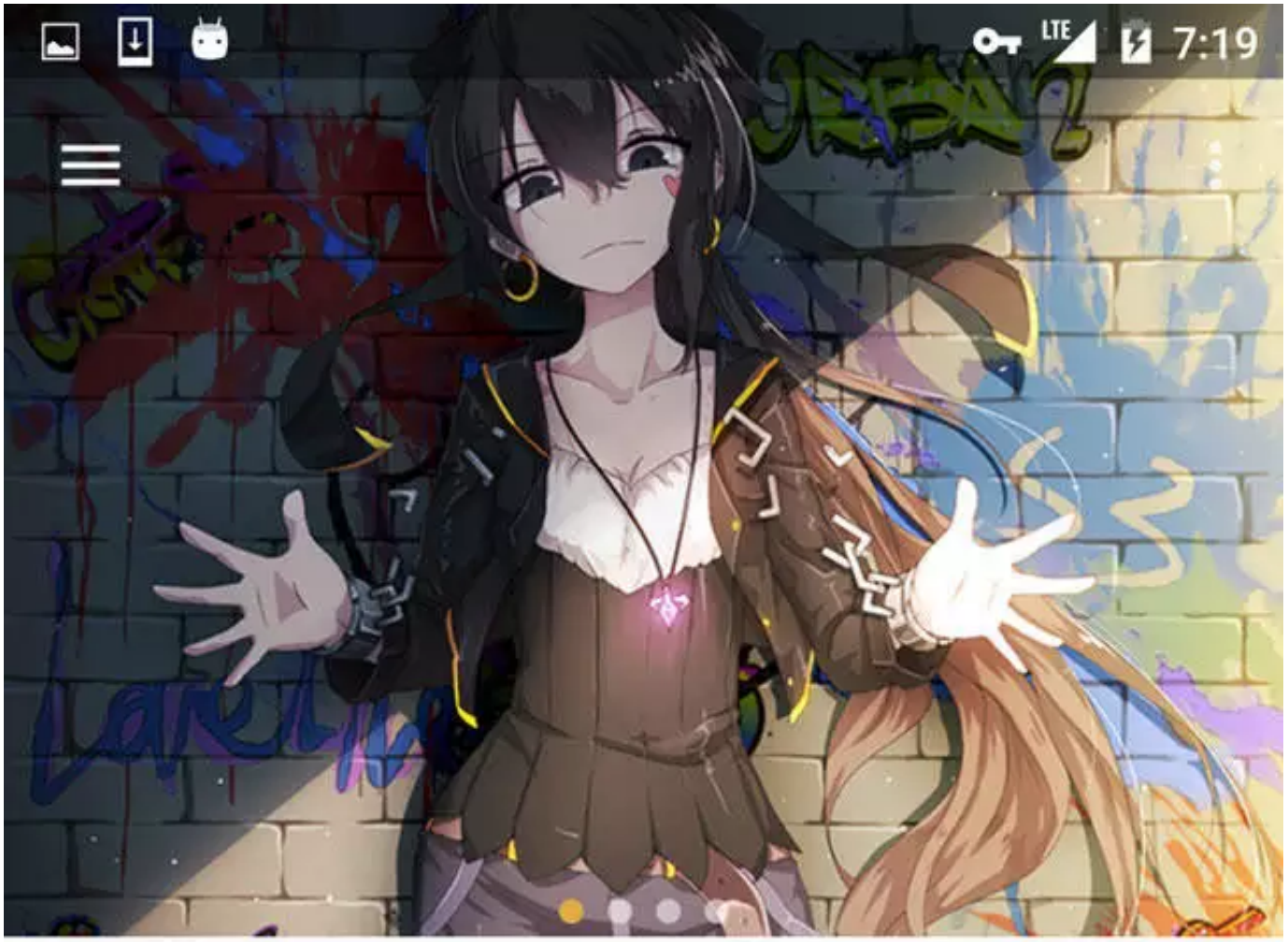
然后这里就有2个问题：

1. Toolbar到了StatusBar的下面(为了凸显问题给Toolbar设了颜色)
2. View在NavigationBar下难以点击

这2个问题也是理所应当就应该存在的。
解决方法：

FitSystemWindowLayout

这里就先说结果吧，使用FitSystemWindowLayout，我为此做了很多适配工作。
这是标准界面：

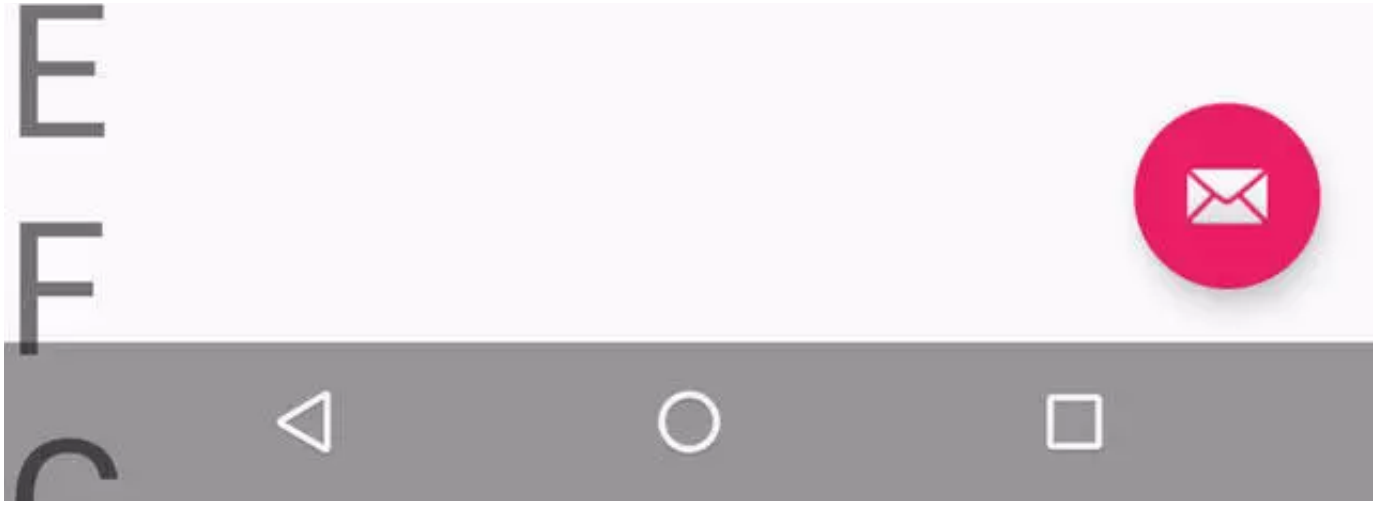


A

B

C

D



QQ图片20160525195401.jpg

这个库提供自动适应StatusBar与NavigationBar的几个Layout。在XML中设置即可。这是上面标准UI的XML：

```
<?xml version="1.0" encoding="utf-8"?>
<com.jude.fitsystemwindowlayout.FitSystemWindowsFrameLayout
    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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:padding_status="false"//不自动腾出StatusBar的空间，为了让图片在StatusBar下绘制
    tools:context="com.jude.demo.MainActivity">

    //这个ViewPager里面放的ImageView
    <com.jude.rollviewpager.RollPagerView
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:rollviewpager_play_delay="3000"/>

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?actionBarSize"
        android:background="#0000"//透明ToolBar
        app:theme="@style/AppTheme.Dark"
```

```
        app:margin_status="true"//让ToolBar去适应StatusBar
         />

    <ScrollView
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:padding_navigation="true"//只对可滑动View有效的属性，自动增加底部内Padding
        android:layout_marginTop="300dp">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/Base.TextAppearance.AppCompat.Display3"
            android:text="A\nB\nC\nD\nE\nF\nG"/>
    </ScrollView>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        app:margin_navigation="true"//让FAB去适应NavigationBar
        android:src="@android:drawable/ic_dialog_email" />

</com.jude.fitsystemwindowlayout.FitSystemWindowsFrameLayout>
```

FitSystemWindow的原理

Android4.4与Android5.0的Insets处理机制完全不同。

Android 5.0的机制：

ViewRootImpl.java中掌管View绘制的函数。

```
private void performTraversals() {
    ......
    dispatchApplyInsets(host);
    ......
```

```
    performMeasure(childWidthMeasureSpec, childHeightMeasureSpec);
    ......
    performLayout(lp, desiredWindowWidth, desiredWindowHeight);
    ......
    performDraw();
}

void dispatchApplyInsets(View host) {
    host.dispatchApplyWindowInsets(getWindowInsets(true /* forceConstruct */));
}
```

SystemBar的尺寸在WindowInsets 中表示出来，比如Insets:(0, 63 , 0, 126)。表示StatusBar高度63，NavigationBar高度126.
dispatchApplyWindowInsets 将WindowInsets 从View树顶部开始分发。

```
//View.java 代码经过精简
public WindowInsets dispatchApplyWindowInsets(WindowInsets insets) {
    if (mListenerInfo != null && mListenerInfo.mOnApplyWindowInsetsListener != null) {
        return mListenerInfo.mOnApplyWindowInsetsListener.onApplyWindowInsets(this, insets);
    } else {
        return onApplyWindowInsets(insets);
    }
}
----------------------------------------------------------------------
//ViewGroup.java
@Override
public WindowInsets dispatchApplyWindowInsets(WindowInsets insets) {
    insets = super.dispatchApplyWindowInsets(insets);
    if (!insets.isConsumed()) {
        final int count = getChildCount();
        for (int i = 0; i < count; i++) {
            insets = getChildAt(i).dispatchApplyWindowInsets(insets);
            if (insets.isConsumed()) {
                break;
            }
        }
    }
    return insets;
}
```

这个方法很简单。在View树中分发Insets，先序遍历。一旦被消费就终止分发。可以看到处理WindowInsets 主要是2个方式。

自定义Insets处理方式的方法1

给目标View注册监听
View接收到Insets。会先判断自己是否被注册了监听，监听是指这个，在这个监听里能够收到Insets。并依据自己情况处理。

```
public void setOnApplyWindowInsetsListener(OnApplyWindowInsetsListener listener) {
    getListenerInfo().mOnApplyWindowInsetsListener = listener;
}
```

自定义Insets处理方式的方法2

重写onApplyWindowInsets
先看看默认的实现。

```
//代码经过精简
public WindowInsets onApplyWindowInsets(WindowInsets insets) {
    if (fitSystemWindowsInt(insets.getSystemWindowInsets())) {
        //如果fitSystemWindowsInt返回true就消耗Instes，好简单的逻辑
        return insets.consumeSystemWindowInsets();
    }
    return insets;
}
```

重点来了fitSystemWindowsInt.它实质性的判断并设置了Padding。

```
private boolean fitSystemWindowsInt(Rect insets) {
    //如果设置了FITS_SYSTEM_WINDOWS这个flag
    if ((mViewFlags & FITS_SYSTEM_WINDOWS) == FITS_SYSTEM_WINDOWS) {
        mUserPaddingStart = UNDEFINED_PADDING;
        mUserPaddingEnd = UNDEFINED_PADDING;
        Rect localInsets = sThreadLocal.get();
        if (localInsets == null) {
            localInsets = new Rect();
            sThreadLocal.set(localInsets);
```

```
        }

        //computeFitSystemWindows主要就是localInsets=insets。并清空insets
        boolean res = computeFitSystemWindows(insets, localInsets);

        mUserPaddingLeftInitial = localInsets.left;
        mUserPaddingRightInitial = localInsets.right;

        //直接应用这个Insets到padding
        internalSetPadding(localInsets.left, localInsets.top,
                localInsets.right, localInsets.bottom);
        return res;
    }
    return false;
}
```

而FITS_SYSTEM_WINDOWS这个flag有2个来源：
代码手动设置：

```
public void setFitsSystemWindows(boolean fitSystemWindows) {
    setFlags(fitSystemWindows ? FITS_SYSTEM_WINDOWS : 0, FITS_SYSTEM_WINDOWS);
}
```

在XML中设置android:fitSystemWindow="true"：

```
case com.android.internal.R.styleable.View_fitsSystemWindows:
    if (a.getBoolean(attr, false)) {
        viewFlagValues |= FITS_SYSTEM_WINDOWS;
        viewFlagMasks |= FITS_SYSTEM_WINDOWS;
    }
    break;
```

这就很明显了。
设置了fitSystemWindow，默认就会消费掉Insets,并设置padding。如果没有设置，会继续遍历直到被某个View消耗。

其实上面的代码是精简后的，实际上对4.4的机制做了一些兼容处理。为了便于理解删掉了。

Android 4.4的机制：

4.4的机制比5.0简单得多。

```
private void performTraversals() {
    ......
    host.fitSystemWindows(mFitSystemWindowsInsets);
    ......
    performMeasure(childWidthMeasureSpec, childHeightMeasureSpec);
    ......
    performLayout(lp, desiredWindowWidth, desiredWindowHeight);
    ......
    performDraw();
}
```

让DecorView执行fitSystemWindows

```
//ViewGroup.java
@Override
protected boolean fitSystemWindows(Rect insets) {
    boolean done = super.fitSystemWindows(insets);
    if (!done) {
        final int count = mChildrenCount;
        final View[] children = mChildren;
        for (int i = 0; i < count; i++) {
            done = children[i].fitSystemWindows(insets);
            if (done) {
                break;
            }
        }
    }
    return done;
}
----------------------------------------------------
//View.java
//与5.0的fitSystemWindowsInt方法一样
protected boolean fitSystemWindows(Rect insets) {
    if ((mViewFlags & FITS_SYSTEM_WINDOWS) == FITS_SYSTEM_WINDOWS) {
        mUserPaddingStart = UNDEFINED_PADDING;
```

```
        mUserPaddingEnd = UNDEFINED_PADDING;
        Rect localInsets = sThreadLocal.get();
        if (localInsets == null) {
            localInsets = new Rect();
            sThreadLocal.set(localInsets);
        }
        boolean res = computeFitSystemWindows(insets, localInsets);
        mUserPaddingLeftInitial = localInsets.left;
        mUserPaddingRightInitial = localInsets.right;
        internalSetPadding(localInsets.left, localInsets.top,
                localInsets.right, localInsets.bottom);
        return res;
    }
    return false;
}
```

分发的过程变到了这里。并且直接就应用了。
与5.0有很大不同(简单了好多)。
重写fitSystemWindows方法即可实现与5.0一样的效果。