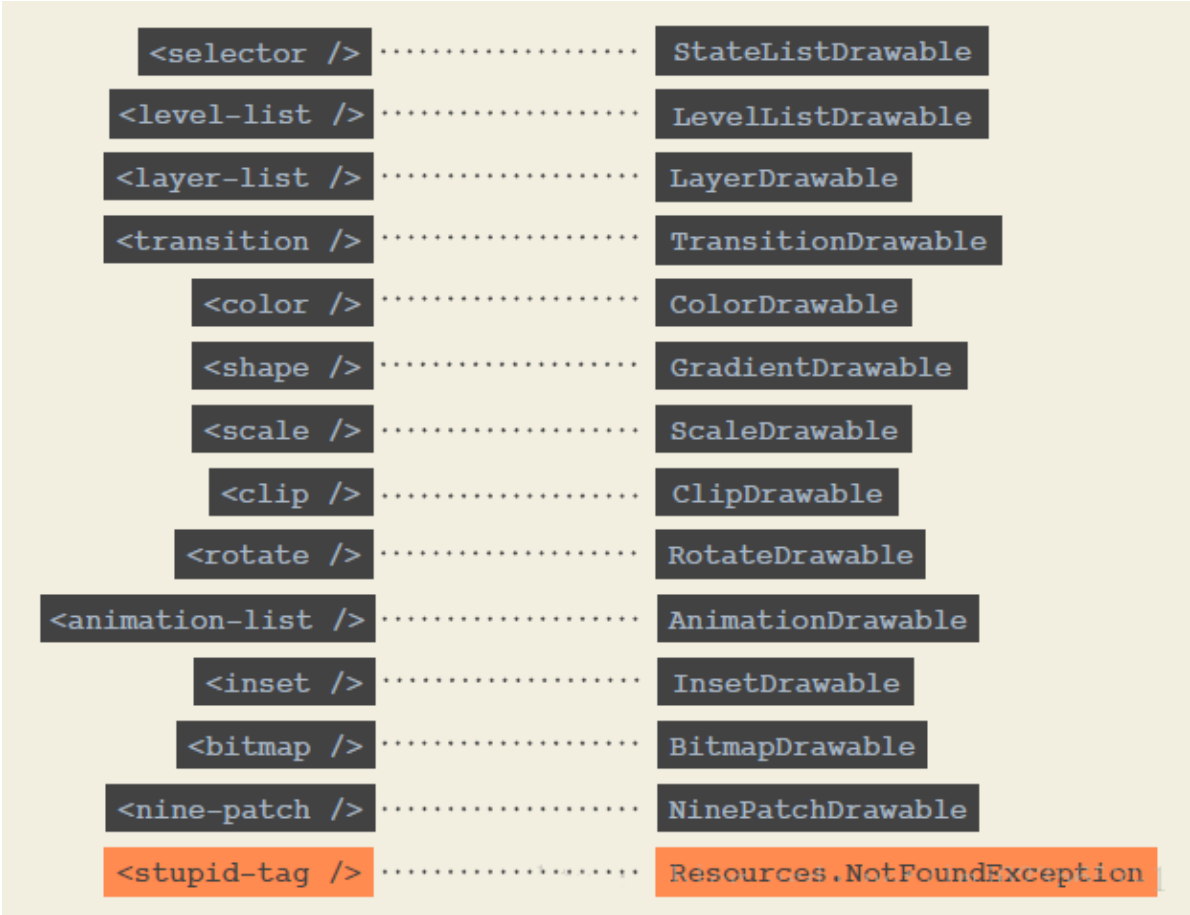
1、概述

Drawable在我们平时的开发中，基本都会用到，而且给大家非常的有用。那么什么是Drawable呢？能够在canvas上绘制的一个玩意，而且相比于View，并不需要去考虑measure、layout，仅仅只要去考虑如何draw（canavs）。当然了，对于Drawable传统的用法，大家肯定不陌生，今天主要给大家带来以下几个Drawable的用法：

1. 自定义Drawable，相比View来说，Drawable属于轻量级的、使用也很简单。以后自定义实现一个效果的时候，可以改变View first的思想，尝试下Drawable first。
2. 自定义状态，相信大家对于State Drawable都不陌生，但是有没有尝试过去自定义一个状态呢？
3. 利用Drawable提升我们的UI Perfermance， 如何利用Drawable去提升我们的UI的性能。

二、Drawable基本概念

一般情况下，除了直接使用放在Drawable下的图片，其实的Drawable的用法都和xml相关，我们可以使用shape、layer-list等标签绘制一些背景，还可以通过selector标签定义View的状态的效果等。当然了基本每个标签都对应于一个真正的实体类，关系如下：



常见的用法这里就不举例了，下面开始看本文的重点。

三、自定义Drawable

关于自定义Drawable，可以通过写一个类，然后继承自Drawable， 类似于自定义View，当然了自定义Drawable的核心方法只有一个，那就是draw。那么自定义Drawable到底有什么实际的作用呢？能干什么呢？

相信大家对于圆角、圆形图片都不陌生，并且我曾经写过通过自定义View实现的方式，具体可参考：

那我今天要告诉你，不需要自定义View，自定义Drawable也能实现，而且更加简单、高效、使用范围更广（你可以作为任何View的背景）。

1、RoundImageDrawable

代码比较简单，下面看下RoundImageDrawable

```
package com.zhy.view;
import android.graphics.Bitmap;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.RectF;
import android.graphics.Shader.TileMode;
import android.graphics.drawable.Drawable;
public class RoundImageDrawable extends Drawable
{
    private Paint mPaint;
    private Bitmap mBitmap;
    private RectF rectF;
    public RoundImageDrawable(Bitmap bitmap)
        {
        mBitmap = bitmap;
        BitmapShader bitmapShader = new BitmapShader(bitmap, TileMode.CLAMP,
                        TileMode.CLAMP);
        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        mPaint.setShader(bitmapShader);
    }
    @Override
        public void setBounds(int left, int top, int right, int bottom)
        {
        super.setBounds(left, top, right, bottom);
        rectF = new RectF(left, top, right, bottom);
    }
    @Override
        public void draw(Canvas canvas)
        {
        canvas.drawRoundRect(rectF, 30, 30, mPaint);
    }
    @Override
        public int getIntrinsicWidth()
        {
        return mBitmap.getWidth();
    }
    @Override
        public int getIntrinsicHeight()
        {
        return mBitmap.getHeight();
    }
    @Override
        public void setAlpha(int alpha)
        {
```

```
        mPaint.setAlpha(alpha);
    }
    @Override
        public void setColorFilter(ColorFilter cf)
        {
        mPaint.setColorFilter(cf);
    }
    @Override
        public int getOpacity()
        {
        return PixelFormat.TRANSLUCENT;
    }
}
```

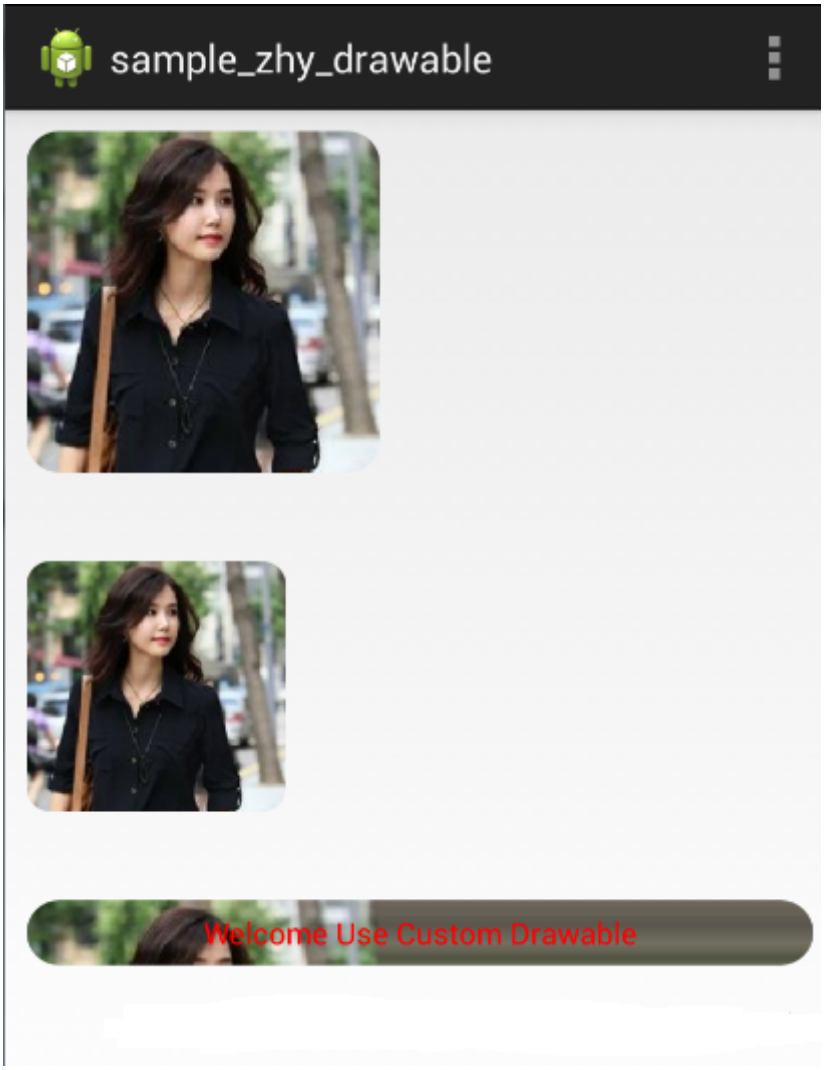
核心代码就是draw了，but，我们只需要一行~~~~setAlpha、setColorFilter、getOpacity、draw这几个方法是必须实现的，不过除了draw以为，其他都很简单。getIntrinsicWidth、getIntrinsicHeight主要是为了在View使用wrap_content的时候，提供一下尺寸，默认为-1可不是我们希望的。setBounds就是去设置下绘制的范围。

ok，圆角图片就这么实现了，easy 不~~

看下用法：

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.mv);
ImageView iv = (ImageView) findViewById(R.id.id_one);
iv.setImageDrawable(new RoundImageDrawable(bitmap));
```

ok，贴一下我们的效果图，两个ImageView和一个TextView



可以看到，不仅仅用于ImageView去实现圆角图片，并且可以作为任何View的背景，在ImageView中的拉伸的情况，配下ScaleType即可。

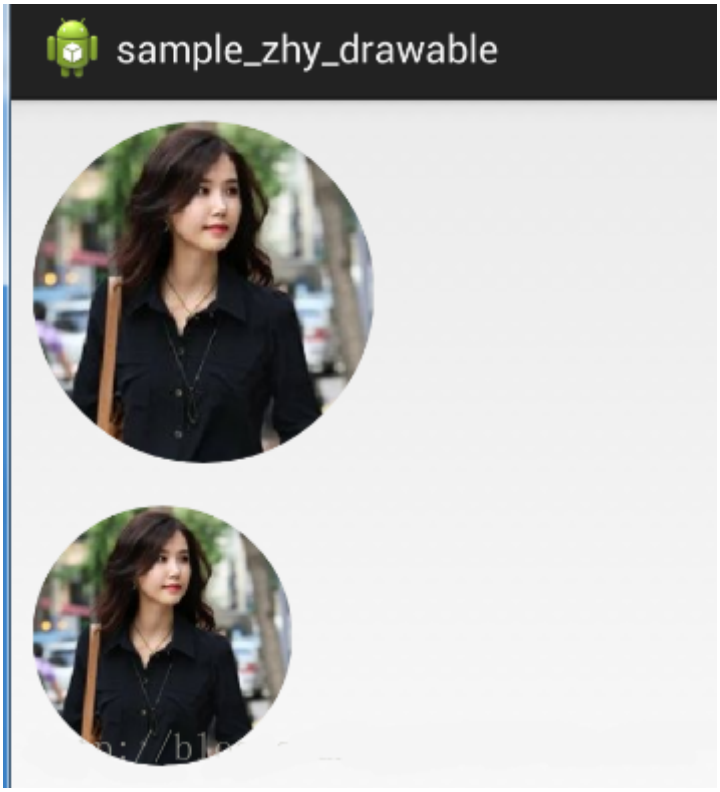
2、CircleImageDrawable

那么下来，我们再看看自定义圆形Drawable的写法:

```
package com.zhy.view;
import android.graphics.Bitmap;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.RectF;
import android.graphics.Shader.TileMode;
import android.graphics.drawable.Drawable;
public class CircleImageDrawable extends Drawable
{
    private Paint mPaint;
    private int mWidth;
    private Bitmap mBitmap ;
    public CircleImageDrawable(Bitmap bitmap)
        {
        mBitmap = bitmap ;
        BitmapShader bitmapShader = new BitmapShader(bitmap, TileMode.CLAMP,
                        TileMode.CLAMP);
```

```
        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        mPaint.setShader(bitmapShader);
        mWidth = Math.min(mBitmap.getWidth(), mBitmap.getHeight());
    }
    @Override
        public void draw(Canvas canvas)
        {
        canvas.drawCircle(mWidth / 2, mWidth / 2, mWidth / 2, mPaint);
    }
    @Override
        public int getIntrinsicWidth()
        {
        return mWidth;
    }
    @Override
        public int getIntrinsicHeight()
        {
        return mWidth;
    }
    @Override
        public void setAlpha(int alpha)
        {
        mPaint.setAlpha(alpha);
    }
    @Override
        public void setColorFilter(ColorFilter cf)
        {
        mPaint.setColorFilter(cf);
    }
    @Override
        public int getOpacity()
        {
        return PixelFormat.TRANSLUCENT;
    }
}
```

一样出奇的简单，再看一眼效果图：



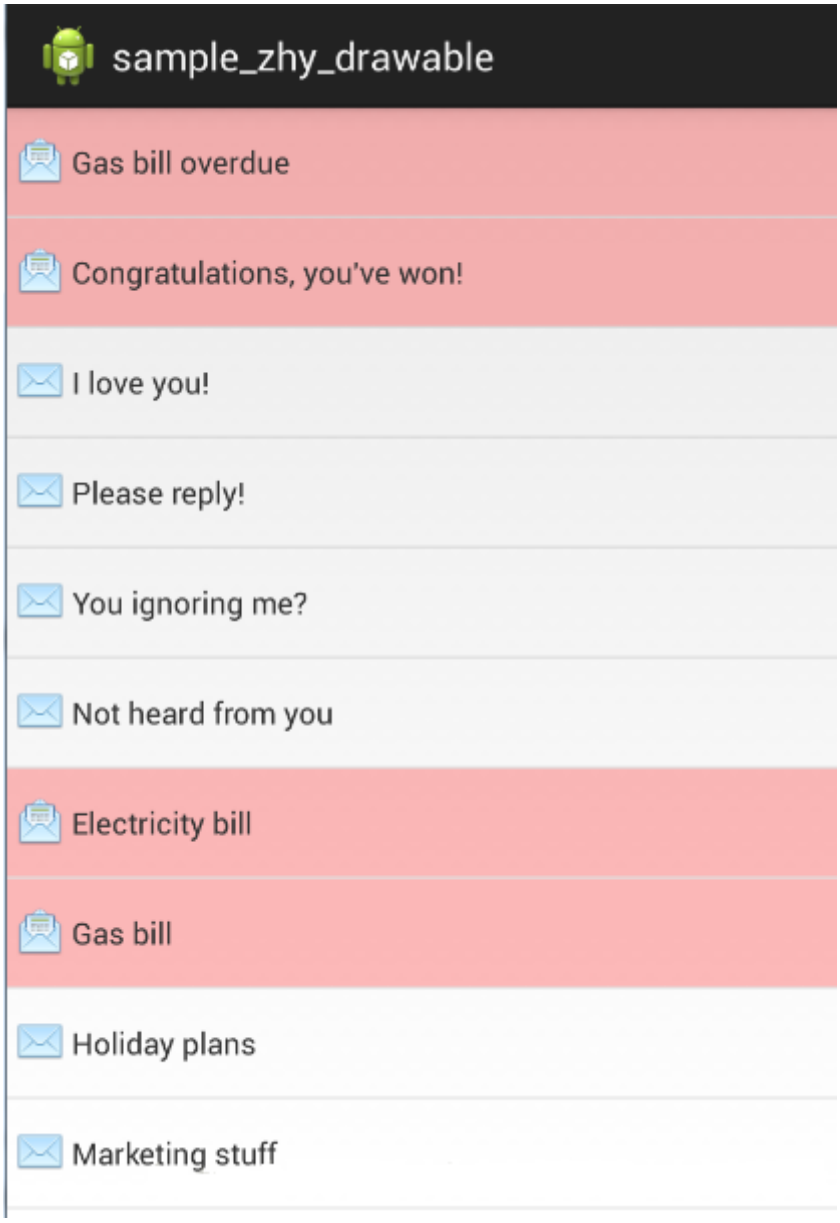
ok，关于自定义Drawable的例子over~~~接下来看自定义状态的。

3、自定义Drawable State

关于Drawable State，state_pressed神马的，相信大家都掌握的特别熟练了。

那么接下来，我们有个需求，类似于邮箱，邮件以ListView形式展示，但是我们需要一个状态去标识出未读和已读：so，我们自定义一个状态state_message_readed。

效果图：



可以看到，如果是已读的邮件，我们的图标是打开状态，且有个淡红色的背景。那么如何通过自定义drawable state 实现呢?

自定义drawable state 需要分为以下几个步骤:

1、res/values/新建一个xml文件：drawable_status.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="MessageStatus">
        <attr name="state_message_readed" format="boolean" />
    </declare-styleable>
</resources>
```

2、继承Item的容器

我们这里Item选择RelativeLayout实现，我们需要继承它，然后复写它的onCreateDrawableState方法，把我们自定义的状态在合适的时候添加进去。

```
package com.zhy.view;
import com.zhy.sample.drawable.R;
import android.content.Context;
import android.util.AttributeSet;
```

```
import android.widget.RelativeLayout;
public class MessageListItem extends RelativeLayout
{
    private static final int[] STATE_MESSAGE_READED = {
R.attr.state_message_readed };
    private Boolean mMessgeReaded = false;
    public MessageListItem(Context context, AttributeSet attrs)
        {
        super(context, attrs);
    }
    public void setMessageReaded(Boolean readed)
        {
        if (this.mMessgeReaded != readed)
                {
            mMessgeReaded = readed;
            refreshDrawableState();
        }
    }
    @Override
        protected int[] onCreateDrawableState(int extraSpace)
        {
        if (mMessgeReaded)
                {
            final int[] drawableState = super
                                .onCreateDrawableState(extraSpace + 1);
            mergeDrawableStates(drawableState, STATE_MESSAGE_READED);
            return drawableState;
        }
        return super.onCreateDrawableState(extraSpace);
    }
}
```

代码不复杂，声明了一个STATE_MESSAGE_READED，然后在mMessgeReaded=true的情况下，通过onCreateDrawableState方法，加入我们自定义的状态。

类似的代码，大家可以看看CompoundButton（CheckBox父类）的源码，它有个checked状态：

```
@Override
    protected int[] onCreateDrawableState(int extraSpace) {
    final int[] drawableState = super.onCreateDrawableState(extraSpace + 1);
    if (isChecked()) {
        mergeDrawableStates(drawableState, CHECKED_STATE_SET);
    }
    return drawableState;
}
```

3、使用

布局文件：

```
<com.zhy.view.MessageListItem
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@drawable/message_item_bg" >
    <ImageView
```

```
        android:id="@+id/id_msg_item_icon"
        android:layout_width="30dp"
        android:src="@drawable/message_item_icon_bg"
        android:layout_height="wrap_content"
        android:duplicateParentState="true"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
      />
    <TextView
        android:id="@+id/id_msg_item_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@id/id_msg_item_icon" />
</com.zhy.view.MessageListItem>
```

很简单，一个图标，一个文本；

Activity

```
package com.zhy.sample.drawable;
import com.zhy.view.MessageListItem;
import android.app.ListActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;
public class CustomStateActivity extends ListActivity
{
    private Message[] messages = new Message[] {
                new Message("Gas bill overdue", true),
                new Message("Congratulations, you've won!", true),
                new Message("I love you!", false),
                new Message("Please reply!", false),
                new Message("You ignoring me?", false),
                new Message("Not heard from you", false),
                new Message("Electricity bill", true),
                new Message("Gas bill", true), new Message("Holiday plans",
false),
                new Message("Marketing stuff", false),
}
;
@Override
    protected void onCreate(Bundle savedInstanceState)
    {
    super.onCreate(savedInstanceState);
    getListView().setAdapter(new ArrayAdapter<Message>(this, -1, messages)
            {
        private LayoutInflater mInflater = LayoutInflater
                            .from(getContext());
        @Override
                    public View getView(int position, View convertView,
ViewGroup parent)
                    {
            if (convertView == null)
```

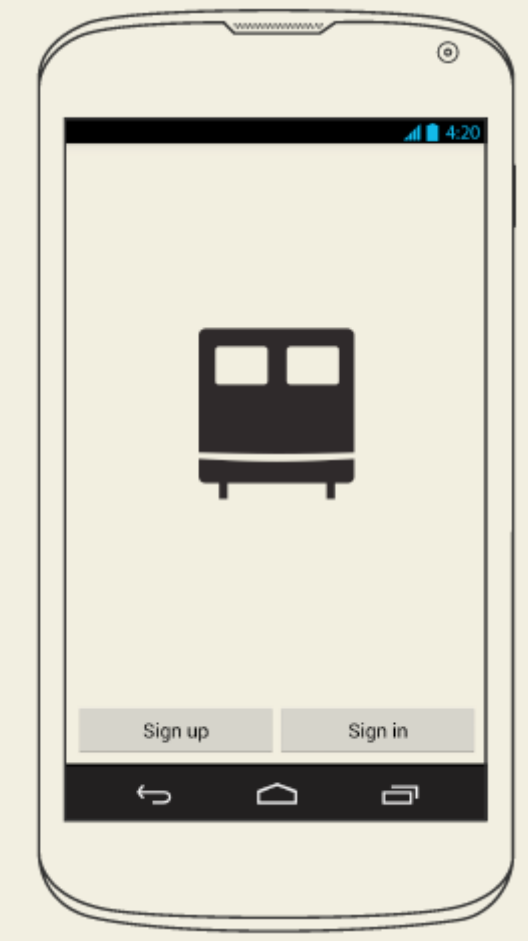
```
                            {
                convertView = mInflater.inflate(R.layout.item_msg_list,
                                                parent, false);
            }
            MessageListItem messageListItem = (MessageListItem) convertView;
            TextView tv = (TextView) convertView
                                    .findViewById(R.id.id_msg_item_text);
            tv.setText(getItem(position).message);
            messageListItem.setMessageReaded(getItem(position).readed);
            return convertView;
        }
    }
    );
}
}
```

代码很简单，但是可以看到，我们需要在getView里面中去使用调用setMessageReaded方法，当然了其他的一些状态，肯定也要手动触发，比如在ACTION_DOWN中触发pressed等。请勿纠结咋没有使用ViewHolder什么的，自己添加下就行。

4、提升我们的UI Perfermance

现在大家越来越注重性能问题，其实没必要那么在乎，但是既然大家在乎了，这里通过Cyril Mottier：master_android_drawables ppt中的一个例子来说明如果利用Drawable来提升我们的UI的性能。

大家看这样一个效果图：



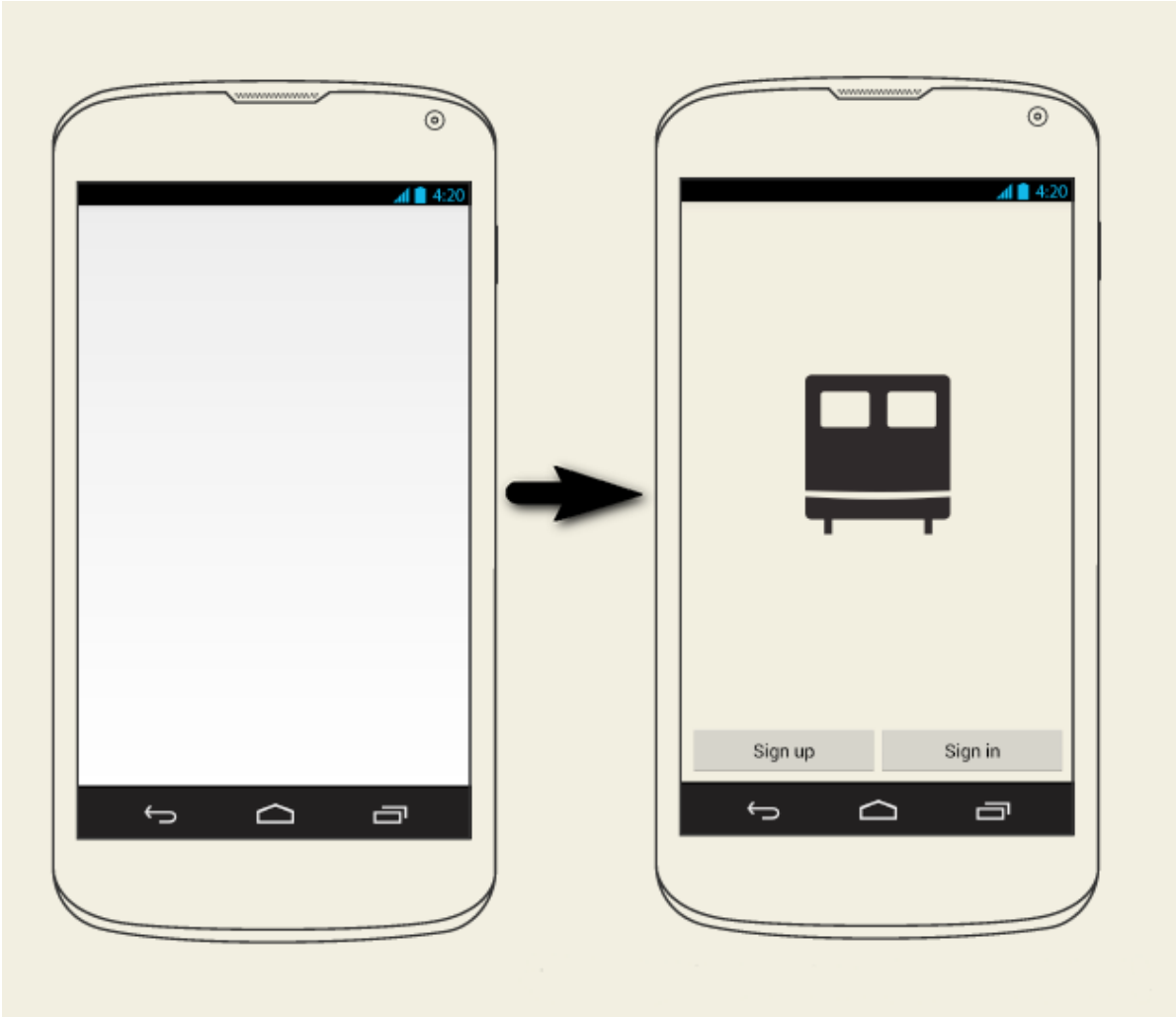
布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/app_background"
    android:padding="8dp" >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginBottom="24dp"
        android:src="@drawable/logo" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:layout_gravity="bottom"
        android:orientation="horizontal" >
        <Button
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="1"
            android:text="@string/sign_up" />
        <Button
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="1"
            android:text="@string/sign_in" />
    </LinearLayout>
</FrameLayout>
```

可以看到最外层是FrameLayout仅仅是为了设置背景图和padding，这样的布局相信很多人也写过。

再看看这个布局作为APP启动时，用户的直观效果：



用户首先看到一个白板，然后显示出我们的页面。接下来，我们将利用Drawable改善我们的UI性能以及用户体验。

1、首先，我们去除我们最外层的FrameLayout，然后自定义一个drawable的xml，叫做logo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item>
        <shape android:shape="rectangle" >
            <solid android:color="@color/app_background" />
        </shape>
    </item>
    <item android:bottom="48dp">
        <bitmap
            android:gravity="center"
            android:src="@drawable/logo" />
    </item>
</layer-list>
```

ok，这个drawable是设置了我们的背景和logo；

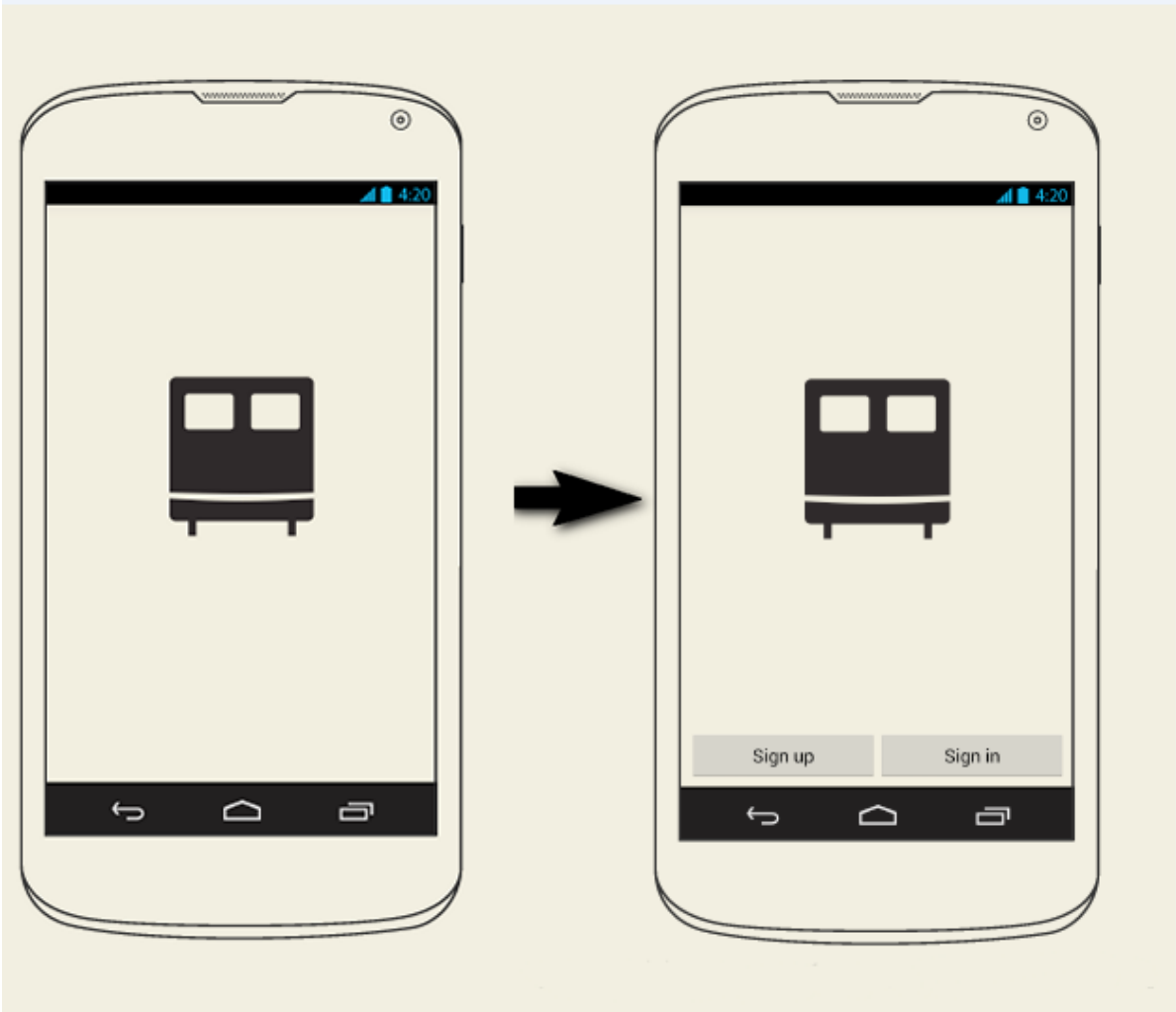
2、将其作为我们当前Activity的windowBackground

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="Theme.Default.NoActionBar"
        parent="@android:style/Theme.Holo.Light.NoActionBar" >
        <item name="android:windowBackground">@drawable/login</item>
    </style>
</resources>
```

3、设置到Activity上：

```
<activity
android:name="LoginActivity"
android:theme="@style/Theme.Default.NoActionBar">
```

Ok，这样不仅最小化了我们的layout，现在我们的layout里面只有一个LinearLayout和两个按钮；并且提升了用户体验，现在用户的直观效果时：



是不是体验好很多，个人很喜欢这个例子~~