第十九期：Android 自定义控件

eoe 特刊

Android 自定义控件

android eoe 开发者门户

优亿市场 Android应用发布与分享平台

目录

【Android 常用基本控件】

1.1 Android 控件基本介绍

Android 本身提供了很多控件，比如我们常用的有：文本控件（TextView 和 EditText）、按钮控件（Button 和 ImageButton）、状态开关按钮（ToggleButton）、单选复选按钮（RadioButton 和 RadioGroup）、单选按钮和复选按钮（CheckBox 和 RadioButton）、图片控件（ImageView）、时钟控件（AnalogClock 和 DigitalClock）、进度条（ProgressBar）和日期与时间选择控件（DatePicker 和 TimePicker）等。但是这些控件并不能满足我们所有的要求。有的时候我们必须要自己定义控件来满足我们的要求。

1.1.1 文本控件（TextView 和 EditText）

在 android 中文本控件分为 ： TextView 控件和 EditText 控件

TextView 控件

TextView 控件继承自 View 类。TextView 控件的功能是向用户显示文本内容，同时可选择性让用户编辑文本。其中 TextView 不允许编辑。

EditText 控件

EditText 控件继承自 TextView。EditText 与 TextView 最大的不同是 EditText 是可以编辑的。

1.1.2 按钮控件（Button 和 ImageButton）

Android 中的按钮可以分为：Button 和 ImageButton 两种控件

Button 控件

Button 控件继承自 TextView 类，Button 的用法比较简单，主要是为 Button 控件设置 View.OnClickListener.监听器，并在监听器的实现代码中编写按钮按下事件的处理代码。

ImageButton 控件

ImageButton 继承自 ImageView。ImageButton 与 Button 最大的区别是 ImageButton 没有 text 属性，既按钮中将显示图片而不是文本。

ImageButton 控件中设置显示图片可以通过 android:src 属性，也可以通过 setImageResurce(int) 方法来实现。

1.1.3 状态开关按钮（ToggleButton）

ToggleButton 控件是继承自 Button。ToggleButton 的状态只能是选中和未选中，并且需要为不同的状态设置不同的显示文本。除了继承自父类的一些属性和方法之外，ToggleButton 也具有一些自己的 ToggleButton 属性。

1.1.4 单选按钮和复选按钮（CheckBox 和 RadioButton）

CheckBox 和 RadioButton 都只有选中和未选中两种状态，可以通过 android：check 属性来设置。不同的是 RadioButton 是单选按钮，需要编制到一个 RadioGroup 中，同一时刻一个 RadioGroup 中只能有一个按钮处于选中状态。

CheckBox 和 RadioButton 都是继承自 CompoundButton 中继承了一些成员

1.1.5 图片控件（ImageView）

ImageView 控件负责显示图片，其图片来源既可以是资源文件的 id，也可以是 Drawable 对象或 Bitmap 对象，还可以是 Content Provider 的 Uri。

1.1.6 时钟控件（AnalogClock 和 DigitalClock）

时钟控件包括 AnalogClock 和 DigitalClock 控件，所不同的是 AnalogClock 继承自 View，而 DigeitalClock 继承自 TextView。并且 AnalogClock 控件显示模拟时钟，只显示时针和分针，而 DigetalClock 显示数字时钟，可精确到秒。

时钟控件比较简单，只需要在布局文件中声明控件即可。

1.1.7 日期与时间选择控件（DatePicker 和 TimePicker）

DatePicker 继承自 FrameLayout 类，日期选择控件的主要功能是向用户提供包含年、月、日的日期数据并允许用户对其进行选择。如果要捕获用户修改日期选择控件中数据的事件，需要为 DatePicker 添加 onDateChangedListener 监听器。

TimePicker 同样继承自 FrameLayout 类。时间选择控件向用户显示一天中的时间（可以为 24 小时制，可以为 AM/PM 制），并允许用户进行选择。如果要捕获用户修改时间数据的事件，便需要为 TimePicker 添加 OnTimeChangedListener 监听器。

1.2 继承已有控件实现自定义控件

通过对 android 本身提供的控件的代码进行研究，android 中控件都是继承 view 类来实现，通过重写 ondraw 方法来绘制我们所需要的控件。通过这个我们得到两点提示，第一，我们可以在已有的控件的基础上，通过重写相关方法来实现我们的需求。第二就是继承 view 类或 viewgroup 类，来绘制我们所需要的控件。一般来讲，通过继承已有的控件来自定义控件要简单一点。下面分别对上述情况举例说明：

1.2.1 控件之 RadioButton

1、举一个最简单的例子说明，比如现在的 RadioButton 按钮只能存在一个 text，如果我们想存储 key-value 对应的键值对，那么我们就需要自定义一个控件。这时定义的控件仅仅比 RadioButton 多了一个存储 key 的控件。实现如下：首先在开始前，我们需要检查在 values 目录下是否有 attrs.xml，如果没有则创建。下面把创建的代码贴出来，如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <declare-styleable name="MRadioButton">
    <attr name="value" format="string"/>
  </declare-styleable>
</resources>
```

其中：外层 name 表示该控件的名称，内层 name 表示属性的名称，format 表示类型。

然后在创建 MRadioButton 类，该类继承 RadioButton。代码如下：

```
public class MRadioButton extends RadioButton implements OnCheckedChangeListener {
        private String mValue;
        public MRadioButton(Context context) {
```

```
        super(context);
      }
      public MRadioButton(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
      }

      public String getValue() {
        return this.mValue;
      }

      public void setValue(String value) {
        this.mValue = value;
      }
      public MRadioButton(Context context, AttributeSet attrs) {
        super(context, attrs);
          TypedArray a = context.obtainStyledAttributes(attrs,
                                R.styleable.MRadioButton);
          this.mValue = a.getString(R.styleable.MRadioButton_value);
          tv.invalidate();
          a.recycle();
        setOnCheckedChangeListener(this);
      }
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    }
}
```

通过 mValue 就可以对该控件设置 key 了，this.mValue = a.getString (R.styleable.MRadioButton_value);可以把在 layout 里面定义的 key 给取出来。下面在看 layout 中的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:eoe="http://schemas.android.com/apk/res/com.eoe.view"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<com.eoe.view.MRadioButton android:id="@+id/mRadioButton" eoe:value="true"

android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="@string/hello" android:textSize="24sp" >
</com.sinxiao.view.MyRadioButton>
</LinearLayout>
```

其中：xmlns:eoe="http://schemas.android.com/apk/res/com.eoe.view"

为定义命名空间路径，在这里定义完后，就可以对 value 进行赋值了。com.eoe.view 为该控件的路径。到此为止已经简单的把如何自定义一个控件的流程讲述一遍。当然这仅仅是最简单的一个例子。如果要实现一些复杂效果，不是仅仅在构造函数中可以完成，

需要重写更多的方法。

1.2.2 组合在一起的 textview 和 editview 控件

我们都知道 view 和 viewGroup 的关系，linearLayout 是继承 viewGroup 没，下面我们分两种情况来介绍如何实现自定义控件。我们要实现一个组合了 textview 和 editview 的控件。第一种的实现不需要在 attrs.xml 中定义，代码如下：

```
public class View1 extends LinearLayout{
    private String Text = "";
     public View1(Context context) {
          this(context, null);
     }
     public View1(Context context, AttributeSet attrs) {
          super(context, attrs);
          int resouceId = -1;
          TextView tv = new TextView(context);
          EditText et = new EditText(context);
          this.setOrientation(LinearLayout.VERTICAL);
          resouceId = attrs.getAttributeResourceValue(null, "Text", 0);
          if (resouceId > 0) {
               Text = context.getResources().getText(resouceId).toString();
          } else {
               Text = "";
          }
          tv.setText(Text);

          addView(tv);
          addView(et, new LinearLayout.LayoutParams

(LayoutParams.FILL_PARENT,LayoutParams.FILL_PARENT));
    }
}
```

布局中的代码如下：

```
    < com.eoe.view.View android:id="@+id/view1"
          android:layout_width="wrap_content"

android:layout_height="wrap_content"
          Text="@string/text1" ></ com.eoe.view.View>
```

第二种的实现如下：首先看下如何在 attrs.xml 中定义：

```
      <declare-styleable name="View2">
          <attr name="Text" format="reference|string"></attr>
          <attr name="Oriental">
               <enum name="Horizontal" value="1"></enum>
               <enum name="Vertical" value="0"></enum>
          </attr>
</declare-styleable>
```

然后看下该类如何写和布局文件中的定义：

```
public class View2    extends LinearLayout{
        public View2(Context context) {
                this(context, null);
            }
            public View2(Context context, AttributeSet attrs) {
                super(context, attrs);
                int resouceId = -1;
                TypedArray typeArray = context.obtainStyledAttributes(attrs,
                        R.styleable.View2);
                TextView tv = new TextView(context);
                EditText et = new EditText(context);
                int N = typeArray.getIndexCount();
                for (int i = 0; i < N; i++) {
                    int attr = typeArray.getIndex(i);
                    switch (attr) {
                    case R.styleable.View2_Oriental:
                        resouceId = typeArray.getInt

(R.styleable.View2_Oriental,
                                0);
                        this.setOrientation(resouceId == 1 ?

LinearLayout.HORIZONTAL
                                : LinearLayout.VERTICAL);
                        break;
                    case R.styleable.View2_Text:
                        resouceId = typeArray.getResourceId(
                                R.styleable.View2_Text, 0);
                        tv.setText(resouceId > 0 ? typeArray.getResources

().getText(
                                resouceId) : typeArray
                                .getString(R.styleable.View2_Text));
                        break;
                    }
                }
                addView(tv);
                addView(et);
                typeArray.recycle();
            }
}
< com.eoe.view.View2
xmlns:eoe="http://schemas.android.com/apk/res/com.eoe.view"
```

```
android:id="@+id/view2"
            android:layout_width="fill_parent"

android:layout_height="wrap_content"
            eoe:Text="eoe" fsms:Oriental="Vertical">
</ com.eoe.view.View2>
```

1.2.3 单独继承 view 类来实现自定义控件

实现一个单独继承 view 类来实现自定义控件，在该方法中，需要重写 ondraw 方法来绘制自己所需要的控件，下面也以一个简单的例子来说明如何实现自定义控件。该方法可以实现所需要的所有的自定义控件。attrs.xml、控件代码和布局代码如下：

```
        <declare-styleable name="myView">
          <attr name="textColor" format="color"/>
          <attr name="textSize" format="dimension"/>
       </declare-styleable>

public class myView extends View{
     private Paint mPaint;

    public myView(Context context) {
          super(context);
    }
    public myView(Context context, AttributeSet attrs) {
          super(context, attrs);
          // TODO Auto-generated constructor stub
           mPaint = new Paint();
           TypedArray a = context.obtainStyledAttributes(attrs,

R.styleable.myView);
           float textSize = a.getDimension(R.styleable.myView_textSize, 30);
           int textColor = a.getColor(R.styleable.myView_textColor, 0x990000FF);


           mPaint.setTextSize(textSize);
           mPaint.setColor(textColor);
           a.recycle();
    }
    public myView(Context context, AttributeSet attrs, int defStyle) {
          super(context, attrs, defStyle);
          // TODO Auto-generated constructor stub
    }
     @Override
     protected void onDraw(Canvas canvas) {
           super.onDraw(canvas);
           canvas.drawRect(new Rect(10 ,10,300,300), mPaint);
```

```
    }
}
< com.eoe.view.myView
    xmlns:eoe="http://schemas.android.com/apk/res/com.eoe.view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    fsms:textSize="100px"
    eoe:textColor="#ABCDEFEF"
/>
```

到此为止，已经把自定义控件的流程和途径简单的介绍完毕。当然，这里所实现的自定义控件都是通过最简单的例子来实现的，起到穿针引线的作用。对于一些复杂的控件的实现也是这样做的。自定义控件不难，需要自己掌握 android 中自定义控件的原理。

1.3 Android 自定义组合控件

目标：实现 textview 和 ImageButton 组合，可以通过 Xml 设置自定义控件的属性。

1.3.1 控件布局

以 Linearlayout 为根布局，一个 TextView，一个 ImageButton。

Xml 代码

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

    < LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android
"

    android:layout_width="fill_parent"
android:layout_height="fill_parent"

    android:gravity="center_vertical">

    < TextView android:layout_height="wrap_content"
android:id="@+id/text1"

    android:layout_width="wrap_content">< /TextView>

    < ImageButton android:layout_width="wrap_content"

    android:layout_height="wrap_content"
android:id="@+id/btn1">< /ImageButton>

    < /LinearLayout>

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

```
    < LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android
"

    android:layout_width="fill_parent"
android:layout_height="fill_parent"

    android:gravity="center_vertical">

    < TextView android:layout_height="wrap_content"
android:id="@+id/text1"

    android:layout_width="wrap_content">< /TextView>

    < ImageButton android:layout_width="wrap_content"

    android:layout_height="wrap_content"
android:id="@+id/btn1">< /ImageButton>

    < /LinearLayout>
```

1.3.2 自定义控件代码，从 LinearLayout 继承

Java 代码

```
public class ImageBtnWithText extends LinearLayout {

    }

     public ImageBtnWithText(Context context) {

     this(context, null);

     }



     public ImageBtnWithText(Context context, AttributeSet attrs) {

     super(context, attrs);


LayoutInflater.from(context).inflate(R.layout.imagebtn_with_text,
this, true);
```

```
    }

    }
```

在构造函数中将 Xml 中定义的布局解析出来。

1.3.3 在主界面布局 xml 中使用自定义控件

Xml 代码

```
< com.demo.widget2.ImageBtnWithText

     android:id="@+id/widget"

     android:layout_width="fill_parent"

     android:layout_height="fill_parent" />

    < com.demo.widget2.ImageBtnWithText

    android:id="@+id/widget"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent" />
```

即使用完整的自定义控件类路径:com.demo.widget2.ImageBtnWithText 定义一个元素。运行可以看到控件已经能够被加载到界面上。

1.3.4 给按钮设置图片和文本

如果图片是固定不变的,那么直接在控件布局中设置 ImageButton 的 src 属性即可。

一、通过 Java 代码设置，在控件代码中提供函数接口：

Java 代码

```
public void setButtonImageResource(int resId) {

     mBtn.setImageResource(resId);

     }
```

```
    public void setTextViewText(String text) {
    mTv.setText(text);
    }
   public void setButtonImageResource(int resId) {
   mBtn.setImageResource(resId);
   }
   public void setTextViewText(String text) {
   mTv.setText(text);
   }
```

然后再在主界面的 onCreate()通过函数调用设置即可。

二、通过 Xml 设置属性

首先定义 Xml 可以设置的属性集合，在 values 下创建 attrs.xml，文件名可随意，一般都叫 attrs.xml

Xml 代码

```
< ?xml version="1.0" encoding="utf-8"?>
  < resources>
    < declare-styleable name="ImageBtnWithText">
    < attr name="android:text"/>
    < attr name="android:src"/>
    < /declare-styleable>
    < /resources
   < ?xml version="1.0" encoding="utf-8"?>
   < resources>
```

```
    < declare-styleable name="ImageBtnWithText">

    < attr name="android:text"/>

    < attr name="android:src"/>

    < /declare-styleable>

    < /resources
```

属性集合名字：ImageBtnWithText，自己可根据实际来定义；

集合中包含的属性列表，每行一个属性。

① 修改自定义控件实现代码，以获取 xml 中定义的属性

Java 代码

```
TypedArray a = context.obtainStyledAttributes(attrs,
R.styleable.ImageBtnWithText);

     CharSequence text =
a.getText(R.styleable.ImageBtnWithText_android_text);

     if(text != null) mTv.setText(text);

     Drawable drawable =
a.getDrawable(R.styleable.ImageBtnWithText_android_src);

     if(drawable != null)
mBtn.setImageDrawable(drawable);

     a.recycle()
```

首先通过 context.obtainStyledAttributes 获得所有属性数组；

然后通过 TypedArray 类的 getXXXX()系列接口获得相应的值。

② 在主界面布局中设置自定义控件的属

android:text="ABC" android:src="@drawable/icon

三、自定义名称属性：

在上述中使用的属性名是 android 系统命名空间的，都以 android 开头，比如 android:text 等。

实际开发中会自定义一些属性名，这些属性名仍然定义在 4.2.1 提到的 attrs.xml 中：

定义属性名

Xml 代码

```
< attr name="appendText" format="string"/>

    < attr name="appendText" format="string"/>
```

和直接使用系统的 attr 不同的是要增加一个 format 属性，说明此属性是什么格式的。format 可选项可参见注 1

使用自定义属性

Xml 代码

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

    < LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"


xmlns:myspace="http://schemas.android.com/apk/res/com.demo.customwidget"

    android:orientation="vertical" android:layout_width="fill_parent"

     android:layout_height="fill_parent">

     < com.demo.widget2.ImageBtnWithText

     android:text="ABC" android:src="@drawable/icon"
android:id="@+id/widget"

     android:layout_width="fill_parent" android:layout_gravity="center"

     android:layout_height="fill_parent" myspace:appendText="123456">

     < /com.demo.widget2.ImageBtnWithText>

     < /LinearLayout>
```

直接在主布局文件中使用此属性 appendText="abc"是不会设置生效的，而是要在主布局 xml 中定义一个 xml 命名空间：

xmlns:myspace="http://schemas.android.com/apk/res/com.demo.customwidget"

命名空间的名字可以自己随便定义，此处为 myspace，即 xmlns:myspace;

后面的地址则有限制，其开始必须为："http://schemas.android.com/apk/res/"，后面则是包名 com.demo.customwidget，

此处的包名与 AndroidManifest.xml 中< manifest>节点的属性 package="com.demo.customwidget"一致，不是自定义控件 Java 代码所在的包，当然简单的程序自定义控件 Java 代码也一般在此包内。

注 1：

注 1：format 可选项

"reference" //引用

"color" //颜色

"boolean" //布尔值

"dimension" //尺寸值

"float" //浮点值

"integer" //整型值

"string" //字符串

"fraction" //百分数,比如 200%

枚举值，格式如下：

< attr name="orientation">

< enum name="horizontal" value="0" />

< enum name="vertical" value="1" />

< /attr>

xml 中使用时：

android:orientation = "vertical"

标志位，位或运算，格式如下：

```
< attr name="windowSoftInputMode">
< flag name = "stateUnspecified" value = "0" />
< flag name = "stateUnchanged" value = "1" />
< flag name = "stateHidden" value = "2" />
< flag name = "stateAlwaysHidden" value = "3" />
< flag name = "stateVisible" value = "4" />
< flag name = "stateAlwaysVisible" value = "5" />
< flag name = "adjustUnspecified" value = "0x00" />
< flag name = "adjustResize" value = "0x10" />
< flag name = "adjustPan" value = "0x20" />
< flag name = "adjustNothing" value = "0x30" />
< /attr>
```

xml 中使用时：

```
android:windowSoftInputMode = "stateUnspecified | stateUnchanged | stateHidden">
```

另外属性定义时可以指定多种类型值,比如：

```
< attr name = "background" format = "reference|color" />
```

xml 中使用时：

```
android:background = "@drawable/图片 ID|#00FF00"
```

1.4 Android 自定义控件----实现 RadioButton 单选附源码

Android 的控件非常漂亮，但不得不承认，也有缺点就是控件功能的弱小。弱小得一个 Radio 只能放一个 text，而没有 value（key）可以存放。使的一个 RadioGroup 里的 RadioButton 可以，同时被选择。

Android 提供的只是一个最基本的控件实现，而非一个完整、强大的实现。可幸的是，

Android 提供了自定义控件的实现。有了自定义控件，我们就可以再 Android 的基础控件上实现我们想要的功能了。

1.4.1 XML 中引用自定义控件

在 XML 中加入自定义控件其实很简单。只需要在控件名字前加入包名即可。如下：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"xmlns:fsms="http://schemas.android.com/apk/res/com.sinxiao.myview" android:orientation="vertical" android:layout_width="fill_parent" android:layout_height="fill_parent" > <com.sinxiao.view.MyRadioGroup
android:id="@+id/radPayOrNot" android:layout_width="wrap_content"
android:layout_height="wrap_content" ><com.sinxiao.view.MyRadioButton
android:id="@+id/isPayDepositTrue"
fsms:value="true" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="@string/yes"
android:textSize="18sp" ></com.sinxiao.view.MyRadioButton><com.sinxiao.view.MyRadioButton android:id="@+id/isPayDepositFalse"
fsms:value="false" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="@string/no"
android:textSize="18sp" ></com.sinxiao.view.MyRadioButton> </com.sinxiao.view.MyRadioGroup></LinearLayout>
```

同样，红色部分可以先不看，也不需要加入到代码中，这个时候加入会报错，请注意。

1.4.2 attrs.xml 属性定义

在我们的思想中，既然我在自定义控件中加入了一个新的属性，那么我就应该能够在 xml 中引用它，并对它赋初始值。我当初也是这样想的。可是却无从下手。就是这一点，折腾了我一个下午。

正解：res/values/attrs.xml 中定义属性，在自定义控件中获取这个属性，然后跟自定义控件的属性相绑定。

attrs.xml 如果没有，就新建一个。这里只存放自定义控件中需要的属性，在我看来，这个文件是一个中介，负责将 layout/xx.xml 里面的对这个变量的引用和自定义控件里面的属性绑定起来。

attrs.xml 完整代码如下：

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

<resources>

<declare-styleable name="RadioButton"><!-- 控件名称-->

   <attr name="value" format="string"/><!-- 属性名称，类型-->

</declare-styleable>

</resources>
```

如果 res 下没有错误的话，在 R 中应该就会生成这些资源的 id。这样我们就能在自定义控件中引用他们。

1.4.3 控件属性与 XML 定义绑定

这下子我们又回到了自定义控件的编写上来了。先看看我们在第一点提到的红色字体部分。这一部分就是实现控件属性与 XML 定义绑定的代码。

```
/**
   * 跟 values/attrs.xml 里面定义的属性绑定
   */
TypedArray a = context.obtainStyledAttributes(attrs,
                    R.styleable.RadioButton);
 this.mValue = a.getString(R.styleable.RadioButton_value);
 a.recycle();
```

TypedArray 其实就是一个存放资源的 Array，首先从上下文中获取到 R.styleable.RadioButton 这个属性资源的资源数组。attrs 是构造函数传进来，应该就是对应 attrs.xml 文件。a.getString(R.styleable.RadioButton_value);这句代码就是获取 attrs.xml 中定义的属性，并将这个属性的值传给本控件的 mValue.最后，返回一个绑定结束的信号给资源：a.recycle()；绑定结束。

1.4.4 在 xml 中对控件赋初始值

综上述，绑定结束后可以在需要赋初始值的地方赋值。

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"xmlns:fsms="http://schemas.android.com/apk/res/com.sinxiao.myview"  android:orientation="vertical"  android:layout_width="fill_parent"  android:layout_height="fill_parent"  >
```

红色部分首先声明命名空间。命名空间为 fsms.路径是 http://schemas.android.com/apk/res/这一部分是不变的，后面接的是 R 的路径:rog.kandy.R。然后在自定义控件的 xml 描述中就可以这样使用 fsms:value="true"。这样就实现了自定义控件的初始化赋值。

1.4.5 RadioGroup、RadioButton 组合控件的实现

上面是自定义控件的实现，下面将要说的是组合控件的实现。在组合控件中，最经常用到的应该就是 RadioGroup 和 RadioButton。 RadioButton 的实现已经在上面介绍了。下面要介绍 RadioGroup 的自定义控件和功能扩展：

代码如下：

```
public class MyRadioGroup extends android.widget.RadioGroup implements
OnCheckedChangeListener {

        private String mValue;

        public MyRadioGroup(Context context, AttributeSet attrs) {

        super(context, attrs);

        this.setOnCheckedChangeListener(this);

        }

        public MyRadioGroup(Context context) {

        super(context);

        this.setOnCheckedChangeListener(this);

        }

        private String tag ="===myRadioGroup";

        // 设置子控件的值

        private void setChildValue(){
```

```
int n = this.getChildCount();
Log.d(tag, "the n is "+n);
for(int i=0;i<n;i++){
  MyRadioButton radio = (MyRadioButton)this.getChildAt(i);
  if(radio.getValue().equals(this.mValue)){
    radio.setChecked(true);
  }else{
    radio.setChecked(false);
  }
}
}
// 获取子类的值
private void getChildValue(){
int n = this.getChildCount();
for(int i=0;i<n;i++){
  MyRadioButton radio = (MyRadioButton)this.getChildAt(i);
  if(radio.isChecked()){
    this.mValue=radio.getValue();
  }
```

```
        }

        }


        public void setTheValue(String value) {

        this.mValue = value;

        }


        public String getTheValue(){

        getChildValue();

        return this.mValue;

        }

       @Override

       public void onCheckedChanged(RadioGroup group, int checkedId) {

                setChildValue();

       }

}
```

RadioGroup 只做两件事：获取子控件（RadioButton）所选择的值；设置子控件要选择的值。

方法非常简单，循环或者 RadioGroup 的子控件，检测哪个控件被 checked，然后 getValue，将此 value 赋值给 RadioGroup 的扩展属性 value。当 MyRadioButton 发生改变的时候，向 RadioGroup 赋值，然后 RadioGroup 根据，value 刷新界面。就实现了单选的效果。

【Android 自定义控件提高篇】

2.1 Android 中自定义属性(attr.xml,TypedArray)的使用

在 **xml** 文件里定义控件的属性，我们已经习惯了 **android:attrs=""** ,那么我们能不能定义自己的属性能，比如:**test:attrs=""** 呢？答案是肯定的. 大致以下步骤:

2.1.1 在 res/values 文件下定义一个 attrs.xml 文件

代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="MyView">
        <attr name="textColor" format="color" />
        <attr name="textSize" format="dimension" />
    </declare-styleable>
</resources>
```

2.1.2 MyView.java 代码修改

如下，其中下面的构造方法是重点，我们获取定义的属性我们 R.sytleable.MyView_textColor, 获取方法中后面通常设定默认值(float textSize = a.getDimension(R.styleable.MyView_textSize, 36);), 防止我们在 xml 文件中没有定义.从而使用默认值!

获取，MyView 就是定义在<declare-styleable name="MyView "></declare-styleable>里的 名字,获取里面属性用 名字_ 属性 连接起来就可以.TypedArray 通常最后调用 .recycle() 方法，为了保持以后使用该属性一致性!

```
public MyView(Context context,AttributeSet attrs)
        {
            super(context,attrs);
            mPaint = new Paint();
            TypedArray a = context.obtainStyledAttributes(attrs,
                    R.styleable.MyView);
            int textColor =
```

```
a.getColor(R.styleable.MyView_textColor,
                        0XFFFFFFFF);
            float textSize =
a.getDimension(R.styleable.MyView_textSize, 36);
            mPaint.setTextSize(textSize);
            mPaint.setColor(textColor);
            a.recycle();
        }
```

MyView.java 全部代码如下:

```
package com.android.tutor;
import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.Paint.Style;
import android.util.AttributeSet;
import android.view.View;
public class MyView extends View {
    private Paint mPaint;
    private Context mContext;
    private static final String mString = "Welcome to Mr Wei's blog";

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }
    public MyView(Context context,AttributeSet attrs)
    {
        super(context,attrs);
        mPaint = new Paint();

        TypedArray a = context.obtainStyledAttributes(attrs,
                R.styleable.MyView);

        int textColor = a.getColor(R.styleable.MyView_textColor,
                0XFFFFFFFF);
        float textSize = a.getDimension(R.styleable.MyView_textSize, 36);
```

```
            mPaint.setTextSize(textSize);
            mPaint.setColor(textColor);

            a.recycle();
        }
        @Override
        protected void onDraw(Canvas canvas) {
            // TODO Auto-generated method stub
            super.onDraw(canvas);
            //设置填充
            mPaint.setStyle(Style.FILL);

            //画一个矩形,前俩个是矩形左上角坐标，后面俩个是右下角坐

            canvas.drawRect(new Rect(10, 10, 100, 100), mPaint);

            mPaint.setColor(Color.BLUE);
            //绘制文字
            canvas.drawText(mString, 10, 110, mPaint);
        }
}
```

2.1.3 将自定义的 MyView 加入布局 main.xml 文件中

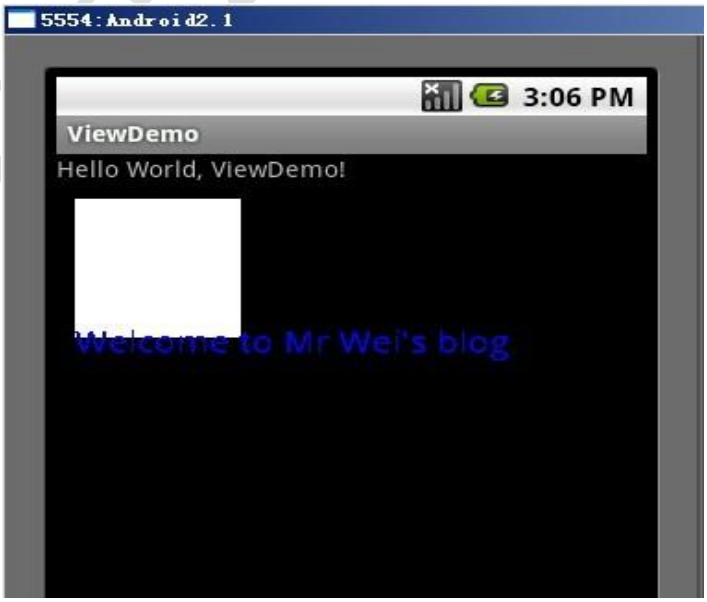
并且使用自定义属性,自定义属性必须加上:

main.xml 全部代码如下:

```
    <?xml

    version="1.0" encoding="utf-8"?>

    <LinearLayout

    xmlns:android="http://schemas.android.com/apk/res/android"



    xmlns:test="http://schemas.android.com/apk/res/com.android.tutor"

        android:orientation="vertical"

        android:layout_width="fill_parent"
```

```
        android:layout_height="fill_parent"
        >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
    <com.android.tutor.MyView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        test:textSize="20px"
        test:textColor="#fff"
    />
    </LinearLayout>
```

2.1.4 运行之效果如下图



2.2 Android 自定义控件外观

在程序开发中，android 系统控件提供的外观往往距离我们要求的有一定差距，此时我们可以通过一些方法来修改。

本文只讨论外观修改，在系统控件上进行功能扩充的自定义控件另外讨论。

首先我们看下系统的 RadioButton：

RadioButton 长成什么样子是由其 Background、Button 等属性决定的，Android 系统使用 style 定义了默认的属性，在 android 源码 android/frameworks/base/core/res/res/values/styles.xml 中可以看到默认的定义：

Xml 代码

```
<style name="Widget.CompoundButton.RadioButton">
        <item name="android:background">@android:drawable/btn_radio_label_background</i
        <item name="android:button">@android:drawable/btn_radio</item>
    </style>
```

即其背景图是 btn_radio_label_background，其 button 的样子是 btn_radio

btn_radio_label_background 是什么？

其路径是 android/frameworks/base/core/res/res/drawable-mdpi/btn_radio_label_background.9.png

可以看到是一个 NinePatch 图片，用来做背景，可以拉伸填充。

btn_radio 是什么？

其路径是 android/frameworks/base/core/res/res/drawable/btn_radio.xml

是个 xml 定义的 drawable，打开看其内容：

Xml 代码:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:state_window_focused="false"
        android:drawable="@drawable/btn_radio_on" />
    <item android:state_checked="false" android:state_window_focused="false"
        android:drawable="@drawable/btn_radio_off" />

    <item android:state_checked="true" android:state_pressed="true"
        android:drawable="@drawable/btn_radio_on_pressed" />
    <item android:state_checked="false" android:state_pressed="true"
        android:drawable="@drawable/btn_radio_off_pressed" />

    <item android:state_checked="true" android:state_focused="true"
        android:drawable="@drawable/btn_radio_on_selected" />
    <item android:state_checked="false" android:state_focused="true"
        android:drawable="@drawable/btn_radio_off_selected" />

    <item android:state_checked="false" android:drawable="@drawable/btn_radio_off" />
    <item android:state_checked="true" android:drawable="@drawable/btn_radio_on" />
</selector>
```

定义了不同状态下 radioButton 长成什么样子。

如果不知道 selector 是什么，就要去看下 Android SDK 文档中 Dev Guide->Application Resources->Resource Types。

以下面一个 item 为例：

<item android:state_checked="true" android:state_pressed="true"
android:drawable="@drawable/btn_radio_on_pressed" />
意思即为当 radiobutton 被选中时，并且被按下时，其 Button 应该长成 btn_radio_on_pressed 这个样子。

文件是 android/frameworks/base/core/res/res/drawable-mdpi/btn_radio_on_pressed.png
drawable 的 item 中可以有以下属性：
android:drawable="@[package:]drawable/drawable_resource"
android:state_pressed=["true" | "false"]
android:state_focused=["true" | "false"]
android:state_selected=["true" | "false"]
android:state_active=["true" | "false"]
android:state_checkable=["true" | "false"]
android:state_checked=["true" | "false"]
android:state_enabled=["true" | "false"]
android:state_window_focused=["true" | "false"]
当按钮的状态和某个 item 匹配后，就会使用此 item 定义的 drawable 作为按钮图片。

从上面分析我们如果要修改 RadioButton 的外观，那么步骤应该是：
(1)制作一个 9patch 的图片作为背景图
准备一副 PNG 图片，其中白色为透明色，是否需要透明各人根据自己需要决定。

运行 SDK/tools/draw9patch
在可伸缩的范围周围加上黑色的线告知系统这些区域可以伸缩。
制作完的图片，周围多了黑色线。

(2)针对不同的状态提供按钮图片
enabled, on: 紫色外框、红色中心点

enabled, off:只有紫色外框

enabled, on, pressed:黄色外框，红色中心点

enabled, off, pressed:黄色外框

disabled, on: 灰色外框、灰色中心点

disabled, off: 灰色外框

其余的状态此处就不再定义。

(3)使用 xml 描述一个 drawable

在 res/drawable/创建 custom_radio_btn.xml

Xml 代码

```
    <?xml version="1.0" encoding="utf-8"?>
    <selector xmlns:android="http://schemas.android.com/apk/res/android">
        <item android:state_enabled="true" android:state_checked="true"
android:state_pressed="true"
              android:drawable="@drawable/enabled_on_pressed" />

        <item android:state_enabled="true" android:state_checked="false"
android:state_pressed="true"
              android:drawable="@drawable/enabled_off_pressed" />

        <item android:state_enabled="true" android:state_checked="true"
              android:drawable="@drawable/enabled_on" />

        <item android:state_enabled="true" android:state_checked="false"
              android:drawable="@drawable/enabled_off" />

        <item android:state_enabled="false" android:state_checked="true"
              android:drawable="@drawable/disabled_on" />

        <item android:state_enabled="false" android:state_checked="false"
              android:drawable="@drawable/disabled_off" />
    </selector>
```

Item 顺序是有讲究的，条件限定越细致，则应该放到前面。比如这儿如果把 1，2 行和 3，4 行的 item 交换，那么 pressed 的就永远无法触发了，因为有 item 已经满足条件返回了。可以理解为代码中的 if 语句。

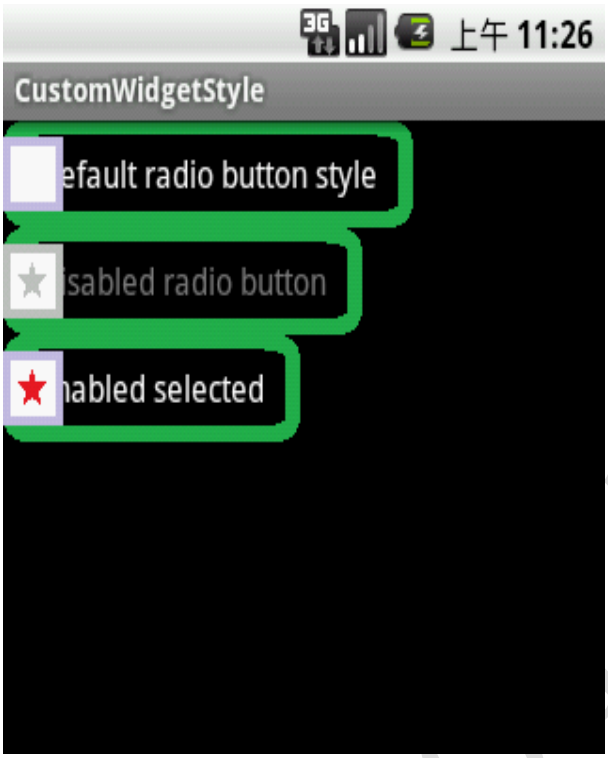
(4)创建一个自定义的 style，并应用到 RaidioButton 的 style 属性上

Xml 代码

```
    <style name="CustomRadioBtn">
            <item name="android:background">@drawable/radio_btn_bg</item>
            <item name="android:button">@drawable/custom_radio_btn</item>
    </style>
```

运行 app 即可看到此 RadioButton 的外观已经改变，此 demo 可以看到文字被按钮遮盖了一部分，

这儿是因第一步制作 9patch 图片时没有留出按钮图片空间来，稍作修改即可。



2.3 Android 自定义动态控件

定义动态控件，支持点击最后一个按钮添加一行控件，否则删除当前选中的该行控件

```
package org.xiangjie.view.blacklist;
import java.util.ArrayList;
import java.util.List;
import org.xiangjie.R;
import android.content.Context;
import android.util.AttributeSet;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class DynamicControl extends LinearLayout {
	private LayoutInflater mInflater;
	private TableLayout layout;
	private int tableRowID;
	private int dynamicBtnID;
	private int dynamicPreID;
	private int dynamicAftID;
```

```
private List<View> idList = new ArrayList<View>();
/**
 * @param context
 */
public DynamicControl(Context context) {
	this(context, null);
}
   public DynamicControl(Context context, AttributeSet attrs) {
	super(context, attrs);
	this.mInflater = LayoutInflater.from(context);
	layout = (TableLayout) mInflater.inflate(R.layout.dynamicvalues, null);
	// 设置默认值
	View view = layout.findViewById(R.id.dynamic_btn_1);
	view.setOnClickListener(getClickListener());
	idList.add(view);
	tableRowID = R.id.dynamic_row_1;
	dynamicBtnID = R.id.dynamic_btn_1;
	dynamicPreID = R.id.dynamic_pre_1;
	dynamicAftID = R.id.dynamic_aft_1;
	addView(layout);
}
private OnClickListener getClickListener() {
	return new OnClickListener() {
		@Override
		public void onClick(View v) {
			clickLogic(v);
		}
	};
}
private void clickLogic(View v) {
	if (!idList.isEmpty()) {
		if (v == idList.get(idList.size() - 1)) {
			addControls(v);
		} else {
			removeControls(v);
		}
	}
}
private TableRow getNewRow() {
	TableRow row = getDefaultTableRow();
	row.addView(getDefaultImageButton());
	row.addView(getDefaultPrevEditText());
	row.addView(getDefaultTextView());
	row.addView(getDefaultAfterEditText());
	return row;
}
```

```
    private void addControls(View v) {
        layout.addView(getNewRow());
        ((ImageButton) v).setImageResource(R.drawable.add_btn);
    }

    private void removeControls(View v) {
        if (idList.size() <= 1) {
            return;
        }
        TableRow parent = (TableRow) v.getParent();
        ((TableLayout) parent.getParent()).removeView(parent);
        idList.remove(v);
    }
    private TableRow getDefaultTableRow() {
        tableRowID++;
        TableRow row = new TableRow(getContext());
        android.widget.TableRow.LayoutParams params = new android.widget.TableRow.Layout
        params.width = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        params.height = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        row.setLayoutParams(params);
        row.setGravity(Gravity.CENTER);
        row.setId(tableRowID);
        return row;
    }
    private EditText getDefaultEditText() {
        EditText text = new EditText(getContext());
        android.widget.TableRow.LayoutParams params = new android.widget.TableRow.Layout
        params.width = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        params.height = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        text.setLayoutParams(params);
        // text.set
        text.setMaxLines(1);
        text.setWidth(80);
        return text;
    }
    private EditText getDefaultPrevEditText() {
        dynamicPreID++;
        EditText text = getDefaultEditText();
        text.setId(dynamicPreID);
        return text;
    }
    private EditText getDefaultAfterEditText() {
        dynamicAftID++;
        EditText text = getDefaultEditText();
        text.setId(dynamicAftID);
        return text;
```

```
    }
    private TextView getDefaultTextView() {
        TextView view = new TextView(getContext());
        android.widget.TableRow.LayoutParams params = new android.widget.TableRow.Layout
        params.width = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        params.height = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        view.setLayoutParams(params);
        view.setText("-");
        return view;
    }
    private ImageButton getDefaultImageButton() {
        dynamicBtnID++;
        ImageButton btn = new ImageButton(getContext());
        btn.setAdjustViewBounds(true);
        android.widget.TableRow.LayoutParams params = new android.widget.TableRow.Layout
        params.width = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        params.height = android.widget.TableRow.LayoutParams.WRAP_CONTENT;
        btn.setLayoutParams(params);
        btn.setImageResource(R.drawable.add_btn);
        btn.setBackgroundResource(R.drawable.add_btn);
        btn.setId(dynamicBtnID);
        btn.setOnClickListener(getClickListener());
        idList.add(btn);
        return btn;
    }
}
```

dynamicvalues.xml

```
view plaincopy to clipboardprint?
<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:id="@+id/dynamic_parent"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TableRow android:id="@+id/dynamic_row_1"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:gravity="center">
        <ImageButton android:src="@drawable/add_btn"
            android:adjustViewBounds="true" android:layout_height="wrap_content"
            android:id="@+id/dynamic_btn_1" android:layout_width="wrap_content"></Ima
        <EditText android:text="" android:id="@+id/dynamic_pre_1"
            android:width="80px" android:layout_width="wrap_content"
            android:maxLines="1" android:layout_height="wrap_content"
            android:numeric="integer"></EditText>
```

```
            <TextView android:text="-" android:layout_width="wrap_content"
                android:layout_height="wrap_content"></TextView>
            <EditText android:text="" android:id="@+id/dynamic_aft_1"
                android:width="80px" android:maxLines="1" android:layout_width="wrap_conter
                android:layout_height="wrap_content" android:numeric="integer"></EditText>
        </TableRow>
</TableLayout>
```

2.4 Android 自定义控件后如何调整自身子控件与父类中子控件的布局

在开发 android 应用的时候，经常要用到一些系统控件，如 AlertDialog，EditTextPreference 等等，但是往往这些系统控件无法完全满足自己需求，于是就要继承这些类，写一个自定义的控件。

图 1

图 2

以这两张图片中显示的自定义控件下面介绍两种布局方法：

方法一：

以 AlertDialog 作为父类的自定义控件为例。

BrightnessSettingDialog.java

```
import android.app.AlertDialog;

 import android.content.Context;

 import android.content.DialogInterface;

 import android.os.Bundle;

 import android.view.LayoutInflater;

 import android.view.View;

 import android.widget.CheckBox;

 import android.widget.ImageView;

 import android.widget.SeekBar;



  public class BrightnessSettingDialog extends AlertDialog implements
DialogInterface

 {



 private SeekBar mSeekBar;

 private CheckBox mCheckBox;

 private ImageView mIcon;

 private int mProgress;



 public BrightnessSettingDialog(Context context) {

 super(context);

 }



 public BrightnessSettingDialog(Context context, int theme) {
```

```
super(context, theme);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
LayoutInflater inflater = (LayoutInflater) getContext().getSystemService(
View view = inflater.inflate(R.layout.brightness_setting_dialog, null);
setView(view);
super.onCreate(savedInstanceState);
mSeekBar = (SeekBar)findViewById(R.id.seekbar);
mCheckBox = (CheckBox) findViewById(R.id.automatic_mode);
mIcon = (ImageView) findViewById(R.id.icon);
mIcon.setImageResource(R.drawable.brightness);
mListener.onBindBrightnessSettingDialog();
}

public SeekBar getSeekBar() {
return mSeekBar;
}

public CheckBox getCheckBox() {
return mCheckBox;
}
}
```

brightness_setting_dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="wrap_content" android:layout_height="wrap_content">
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/screen" android:layout_width="fill_parent"
android:layout_height="wrap_content" android:orientation="vertical">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent" android:layout_height="wrap_content"
android:orientation="vertical" android:gravity="center_horizontal"
android:paddingBottom="10dip">
<ImageView android:id="@+id/icon" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:paddingTop="10dip" />

<CheckBox android:id="@+id/automatic_mode"
android:layout_width="fill_parent" android:layout_height="wrap_content"
android:text="@string/automatic_brightness"
android:textAppearance="?android:attr/textAppearanceSmall"
android:layout_marginTop="6dip" android:layout_marginLeft="10dip"
android:layout_marginRight="20dip" />

<SeekBar android:id="@+id/seekbar" android:layout_width="fill_parent"
android:layout_height="wrap_content" android:padding="10dip" />
</LinearLayout>
</ScrollView>
</RelativeLayout>
```

首先创建 alertdialog 的子类，在 OnCreate 方法中通过 LayoutInflater 来加载控件：

```
LayoutInflater inflater = (LayoutInflater) getContext().getSystemService(
                        Context.LAYOUT_INFLATER_SERVICE);
                View view = inflater.inflate(R.layout.brightness_setting_dialog, null);
```

接着调用 setView 方法，将加载的控件添加到父控件中，最后调用父类的 onCreate 方法即可。

要注意的是，xml 中采用的是 RelativeLayout 布局。

方法二：

方法一中的 AlertDialog 的子类，确定和取消按钮由父类创建的，并且在布局中固定在整个控件的最底端，因此可以通过 inflate RelativeLayout 的方式来创建自定义控件，但是图二所示，如果这个控件中的编辑框是由父类创建的，那么如何在子类中控制这个编辑框恰好放在想 放的位置呢？下以 EditTextPreference 作为父类的自定义控件为例实现这个自定义控件。

代码摘要：

EditPhoneNumberPreference.java

```
...

 public class EditPhoneNumberPreference extends
EditTextPreference {

 …

 /**

 * Overriding EditTextPreference's onAddEditTextToDialogView.

 *

 * This method attaches the EditText to the container specific
to this

 * preference's dialog layout.

 */

 @Override

 protected void onAddEditTextToDialogView(View dialogView,
EditText editText) {



 // look for the container object

 ViewGroup container = (ViewGroup) dialogView

 .findViewById(R.id.edit_container);



 // add the edittext to the container.

 if (container != null) {

 container.addView(editText,
ViewGroup.LayoutParams.FILL_PARENT,

 ViewGroup.LayoutParams.FILL_PARENT);
```

```
}

}

...

}
```

pref_dialog_editphonenumber.xml

```
<?xml version="1.0" encoding="utf-8"?>
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:padding="5dip"
 android:orientation="vertical">

 <TextView android:id="@+android:id/message"
 style="?android:attr/textAppearanceMedium"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:textColor="?android:attr/textColorPrimary"
 android:paddingLeft="10dip"
 android:paddingRight="10dip"/>

 <LinearLayout
 android:id="@+id/number_field"
 android:layout_width="fill_parent"
 android:layout_height="0dip"
 android:layout_weight="1.0"
 android:addStatesFromChildren="true"
 android:gravity="center_vertical"
 android:baselineAligned="false"
 android:paddingLeft="10dip"
 android:paddingRight="10dip">

 <LinearLayout android:id="@+id/edit_container"
 android:layout_width="0dip"
 android:layout_weight="1"
 android:layout_height="wrap_content"/>

 <ImageButton android:id="@+id/select_contact"
 android:layout_width="wrap_content"
 android:layout_height="fill_parent"
 android:padding="10dip"
 android:src="@drawable/ic_button_contacts" />
```

```
</LinearLayout>

</LinearLayout>
```

继承 EditTextPreference，并实现 onAddEditTextToDialogView 方法，这个方法会在 onBindDialogView 中调用，而 onBindDialogView 方法则会在父类显示 dialog 控件时调用，同时父类 EditTextPreference 会将自己创建的 editText 指针传递下来。

于是可以在 onAddEditTextToDialogView 方法中创建一个容器和父类的 editText 绑定：

```
// look for the container object

   ViewGroup container = (ViewGroup) dialogView

 .findViewById(R.id.edit_container);



 // add the edittext to the container.

   if (container != null) {

 container.addView(editText,
ViewGroup.LayoutParams.FILL_PARENT,

 ViewGroup.LayoutParams.FILL_PARENT);

 }
```

然后在布局文件中将绑定的 container 按照正确的方式布局即可：

```
<LinearLayout
 android:id="@+id/number_field"
 android:layout_width="fill_parent"
 android:layout_height="0dip"
 android:layout_weight="1.0"
 android:addStatesFromChildren="true"
 android:gravity="center_vertical"
 android:baselineAligned="false"
 android:paddingLeft="10dip"
 android:paddingRight="10dip">

 <LinearLayout android:id="@+id/edit_container"
 android:layout_width="0dip"
 android:layout_weight="1"
 android:layout_height="wrap_content"/>
```

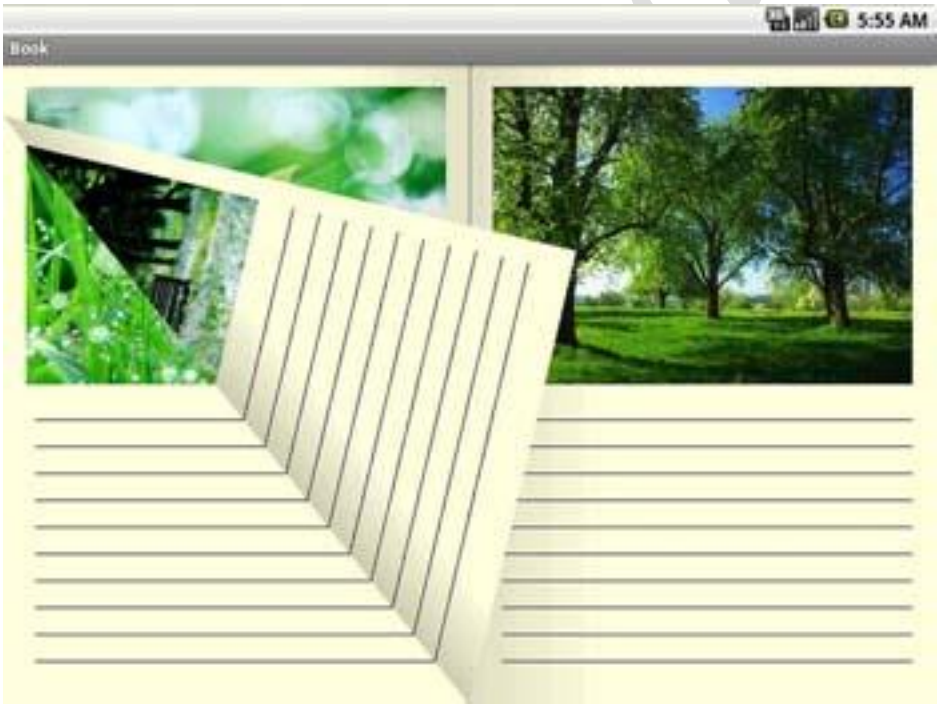
```
<ImageButton android:id="@+id/select_contact"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:padding="10dip"
android:src="@drawable/ic_button_contacts" />
</LinearLayout>
```

方法一相对简单，但是不能灵活调整布局，方法二需要自己实现方法将父类子控件的指针传递给子类，相对较麻烦，一般在实际开发当中都是通过两种方法结合在一起灵活使用。

【Android 自定义控件实例教程】

3.1 Android 自定义控件 eBook 翻书效果

效果图：



Book.java 文件：

```
package com.book;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
public class Book extends Activity {
    /** Called when the activity is first created. */
 eBook mBook;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mBook = (eBook)findViewById(R.id.my_book);
        mBook.addLayoutRecForPage(R.layout.page,21);
        mBook.setListener(new eBook.Listener() {
  public void onPrevPage() {
   updateContent();
  }
  public void onNextPage() {
   updateContent();
  }
  public void onInit() {
   updateContent();
  }
 });
    }

    private void updateContent(){
     int index = mBook.getIndexForLeftPage();
     View left = mBook.getLeftPage(),right = mBook.getRightPage();
        View next1 = mBook.getNextPage1(),next2 = mBook.getNextPage2();
        View prev1 = mBook.getPrevPage1(),prev2 = mBook.getPrevPage2();
        if(left != null)setImg(left,index);
        if(right != null)setImg(right,index+1);
        if(next1 != null)setImg(next1,index+2);
        if(next2 != null)setImg(next2,index+3);
        if(prev1 != null)setImg(prev1,index-1);
        if(prev2 != null)setImg(prev2,index-2);
        mBook.invalidate();
    }

 private void setImg(View v , int index){
  if(index >= 0 && index < 20){
  ImageView img = (ImageView)v.findViewById(R.id.book_img);
  if(img == null)return;
  Log.d("eBook","set Img");
  switch(index%6){
  case 0:
```

```
   img.setImageResource(R.drawable.p1);
   break;
  case 1:
   img.setImageResource(R.drawable.p2);
   break;
  case 2:
   img.setImageResource(R.drawable.p3);
   break;
  case 3:
   img.setImageResource(R.drawable.p4);
   break;
  case 4:
   img.setImageResource(R.drawable.p5);
   break;
  case 5:
   img.setImageResource(R.drawable.p6);
   break;
  default:
   break;
   }
  }
 }
}
```

main.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
 <com.book.eBook android:id="@+id/my_book"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
</LinearLayout>
```

page.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="20dip"
    android:background="#FFFFDD">
    <ImageView android:layout_width="fill_parent" android:id="@+id/book_img"
    android:layout_height="fill_parent" android:layout_weight="1"
    android:scaleType="fitXY" android:src="http://wallage.blog.163.com/blog/@drawable/p1"/>
```

```
    <com.book.TelEdit
    android:id="@+id/book_text"
    android:layout_width="fill_parent"
    android:background="#ffffdd"
    android:gravity="top"
    android:typeface="sans"
    android:capitalize="sentences"
    android:lineSpacingExtra="5dip"
    android:textSize="15dip"
    android:textColor="#000000"
    android:layout_height="fill_parent"
    android:paddingTop="30dip"
    android:layout_weight="1"/>
</LinearLayout>
```

控件 TelEdit.java 代码：

```
package com.book;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.WindowManager;
import android.widget.EditText;
public class TelEdit extends EditText {
 Context mContext;
 public TelEdit(Context context) {
  super(context);
  mContext = context;
 }
 public TelEdit(Context context, AttributeSet attrs) {
  super(context, attrs);
  mContext = context;
 }
 public TelEdit(Context context, AttributeSet attrs, int defStyle) {
  super(context, attrs, defStyle);
  mContext = context;
 }
 protected void onDraw(Canvas canvas) {
  WindowManager wm = (WindowManager) mContext.getSystemService("window");
  int windowWidth = wm.getDefaultDisplay().getWidth();
  int windowHeight = wm.getDefaultDisplay().getHeight();
  Paint paint = new Paint();
  paint.setStyle(Paint.Style.FILL);
  paint.setColor(Color.BLACK);
```

```
 int paddingTop = getPaddingTop();
 int paddingBottom = getPaddingBottom();
 int scrollY = getScrollY();
 int scrollX = getScrollX() + windowWidth;
 int innerHeight = scrollY + getHeight() - paddingBottom;
 int lineHeight = getLineHeight();
 int baseLine = scrollY
  + (lineHeight - ((scrollY - paddingTop) % lineHeight));
 int x = 8;
 while (baseLine < innerHeight) {
  canvas.drawLine(x, baseLine, scrollX - x, baseLine, paint);
  baseLine += lineHeight;
 }
 super.onDraw(canvas);
}
}
```

eBook.java 文件部分代码：

```
package com.book;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Point;
import android.graphics.PorterDuffXfermode;
import android.graphics.Shader;
import android.graphics.PorterDuff.Mode;
import android.util.AttributeSet;
import android.util.Log;
import android.view.GestureDetector;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.GestureDetector.OnGestureListener;
import android.widget.FrameLayout;
import android.widget.LinearLayout;

public class eBook extends FrameLayout{
```

```
public static final String LOG_TAG = "eBook";
List<Integer> myRecPages;
int totalPageNum;
Context mContext;
boolean hasInit = false;
final int defaultWidth = 600 , defaultHeight = 400;
int contentWidth = 0;
int contentHeight = 0;
View leftPage,rightPage,llPage,lrPage,rrPage,rlPage;
LinearLayout mView;
bookView mBookView;
boolean closeBook = false;

private enum Corner {
 LeftTop,
 RightTop,
 LeftBottom,
 RightBottom,
 None
};
private Corner mSelectCorner;
final int clickCornerLen = 250*250; //50dip
float scrollX = 0,scrollY = 0;
int indexPage = 0;

private enum State {
 ABOUT_TO_ANIMATE,
 ANIMATING,
 ANIMATE_END,
 READY,
 TRACKING
};
private State mState;
private Point aniStartPos;
private Point aniStopPos;
private Date aniStartTime;
private long aniTime = 2000;
private long timeOffset = 900;

Listener mListener;

private GestureDetector mGestureDetector;
private BookOnGestureListener mGestureListener;

public eBook(Context context) {
 super(context);
```

```
 Init(context);
 }
 public eBook(Context context, AttributeSet attrs) {
  super(context, attrs);
  Init(context);
 }
...省略
}
```

该控件大致实现方法：

eBook 继承 FrameLayout，好处在于 FrameLayout 有图层效果，后添加的 View 类能覆盖前面的 View。

初始化：定义一个 LinearLayout 的成员变量 mView，将 page.xml inflate 成 View 分别用 leftPage，rightPage 引用，并初始化其数据，将 leftPage，rightPage 通过 addView 添加到 mView，然后将 mView 添加到 eBook。在 eBook 里定义一个私有类 BookView extends View。 并定义成员变量 BookView mBookView； 最后将 mBookView 添加的 eBook 中，这样，mView 中的内容为书面内容，mBookView 中的内容为特效内容。

后续手势动作：可将各种手势的特效动作画于 mBookView 的画布中。

3.2 Android 控件源码剖析

关于自定义 android 空间网上的例子也很多，但是大都是一些比较皮毛的例子，在离实际项目开发还有一段距离。但是要想把自定义空间学透彻，怎么办呢？当然，就是从 android 本身自带的控件学起，解剖它的原理和重点。所以我现在来给大家解剖一下 android 自带控件的源码。

先看 android 的 Button 控件源码，这个源码比较简单。

```
16
17 package android.widget;
18
19 import android.content.Context;
20 import android.util.AttributeSet;
21 import android.util.Log;
22 import android.view.MotionEvent;
23 import android.view.KeyEvent;
24 import android.widget.RemoteViews.RemoteView;
25
26
28 * Represents a push-button widget. Push-buttons can be
97 @RemoteView
98 public class Button extends TextView {
99     public Button(Context context) {
100         this(context, null);
101     }
102
103     public Button(Context context, AttributeSet attrs) {
104         this(context, attrs, com.android.internal.R.attr.buttonStyle);
105     }
106
107     public Button(Context context, AttributeSet attrs, int defStyle) {
108         super(context, attrs, defStyle);
109     }
110 }
111
```

这个 Button 的源代码看起来有一百多行，绝大部分是注释，只有几行代码有用。就是有三个构造函数，而且重要的也只是第二个构造函数，第三个构造函数也是继承父类的构造函数。我们再看一个 ImageButton 的源码。

```
18
19 import android.content.Context;
20 import android.os.Handler;
21 import android.os.Message;
22 import android.util.AttributeSet;
23 import android.view.MotionEvent;
24 import android.widget.RemoteViews.RemoteView;
25
26 import java.util.Map;
27
29 * <p>
74 @RemoteView
75 public class ImageButton extends ImageView {
76     public ImageButton(Context context) {
77         this(context, null);
78     }
79
80     public ImageButton(Context context, AttributeSet attrs) {
81         this(context, attrs, com.android.internal.R.attr.imageButtonStyl
82     }
83
84     public ImageButton(Context context, AttributeSet attrs, int defStyle
85         super(context, attrs, defStyle);
86         setFocusable(true);
87     }
88
89     @Override
90     protected boolean onSetAlpha(int alpha) {
91         return false;
92     }
93 }
```

从代码可以看出，和刚才的 Button 源码很相似。也是有三个构造函数，再加上一个重写父类的 onSetAlpha 方法，从这个方法看出 ImageButton 不可以设置透明度了，因为它直接返回的是 false。我在这里就先只说这两个空间，其实其它的也很类似，都是只有几个构造函数，然后加上一些重新父类的方法。所以重点还是他们各自的父类。下面我就简单说下他们的父类代码。由于父类的代码比较多，我也只是抽些重要的说说。

```
6  */
7  @RemoteView
8  public class ImageView extends View {
9      // settable by the client
0      private Uri mUri;
1      private int mResource = 0;
2      private Matrix mMatrix;
3      private ScaleType mScaleType;
4      private boolean mHaveFrame = false;
5      private boolean mAdjustViewBounds = false;
6      private int mMaxWidth = Integer.MAX_VALUE;
7      private int mMaxHeight = Integer.MAX_VALUE;
```

```
@RemoteView
public class TextView extends View implements ViewTreeObserve
    static final String LOG_TAG = "TextView";
    static final boolean DEBUG_EXTRACT = false;

    private static int PRIORITY = 100;

    final int[] mTempCoords = new int[2];
    Rect mTempRect;

    private ColorStateList mTextColor;
    private int mCurTextColor;
    private ColorStateList mHintTextColor;
    private ColorStateList mLinkTextColor;
    private int mCurHintTextColor;
    private boolean mFreezesText;
    private boolean mFrozenWithFocus;
    private boolean mTemporaryDetach;
    private boolean mDispatchTemporaryDetach;
```

从中看出，他们的父类都会继承 View 类。这个很强大，怎么个强大法呢？在 Android 中，几乎所有能看到的元素（控件）都继承自 View 类。通过继承 View，可以很方便地定制出有个性的控件出来。

```
    public TextView(Context context) {
        this(context, null);
    }

    public TextView(Context context,
                    AttributeSet attrs) {
        this(context, attrs, com.android.internal.R.attr.textViewStyle);
    }

    @SuppressWarnings("deprecation")
    public TextView(Context context,
                    AttributeSet attrs,
                    int defStyle) {
        super(context, attrs, defStyle);
        mText = "";

         = new TextPaint(Paint.ANTI_ALIAS_FLAG);
        mTextPaint.density = getResources().getDisplayMetrics().density;
        mTextPaint.setCompatibilityScaling(
```

这个代码是不是感觉好像在那里看到过？对，就是在上面的 Button 和 ImageButton 里面。他们里面覆盖的就是这些构造函数。在这个构造函数和一般的构造函数也差不多，都是一些初始化工作。

```
public boolean onPreDraw() {
    if (mPreDrawState != PREDRAW_PENDING) {
        return true;
    }

    if (mLayout == null) {
        assumeLayout();
    }

    boolean changed = false;

    SelectionModifierCursorController selectionControl
    if (mSelectionModifierCursorController != null) {
        selectionController = (SelectionModifierCursor
            mSelectionModifierCursorController;
    }
```

这是绘画之前的预处理

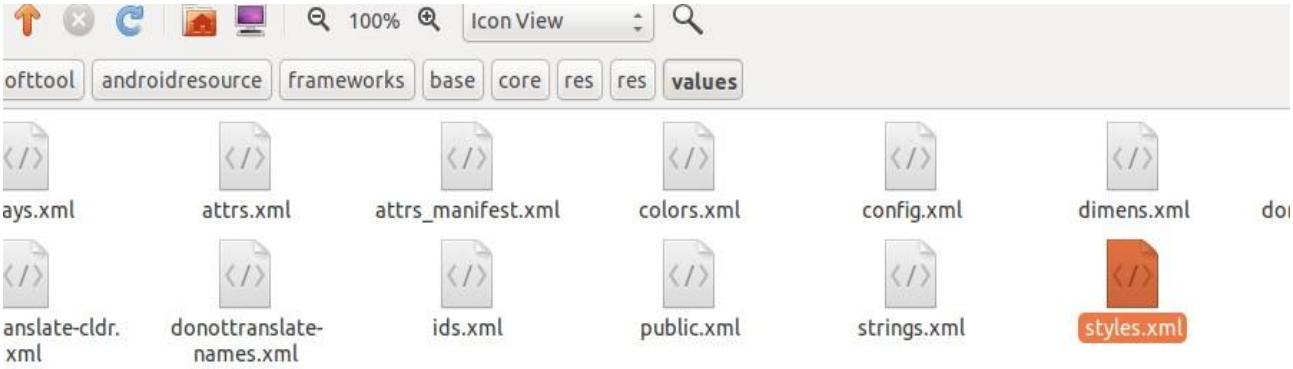
```
@Override
protected void onDraw(Canvas canvas) {
    restartMarqueeIfNeeded();

    // Draw the background for this view
    super.onDraw(canvas);

    final int compoundPaddingLeft = getCompoundPaddingLeft();
    final int compoundPaddingTop = getCompoundPaddingTop();
    final int compoundPaddingRight = getCompoundPaddingRight();
    final int compoundPaddingBottom = getCompoundPaddingBottom();
    final int scrollX = mScrollX;
    final int scrollY = mScrollY;
    final int right = mRight;
    final int left = mLeft;
    final int bottom = mBottom;
    final int top = mTop;

    final Drawables dr = mDrawables;
    if (dr != null) {
        /*
         * Compound, not extended, because the icon is not clipped
         * if the text height is smaller.
         */
```

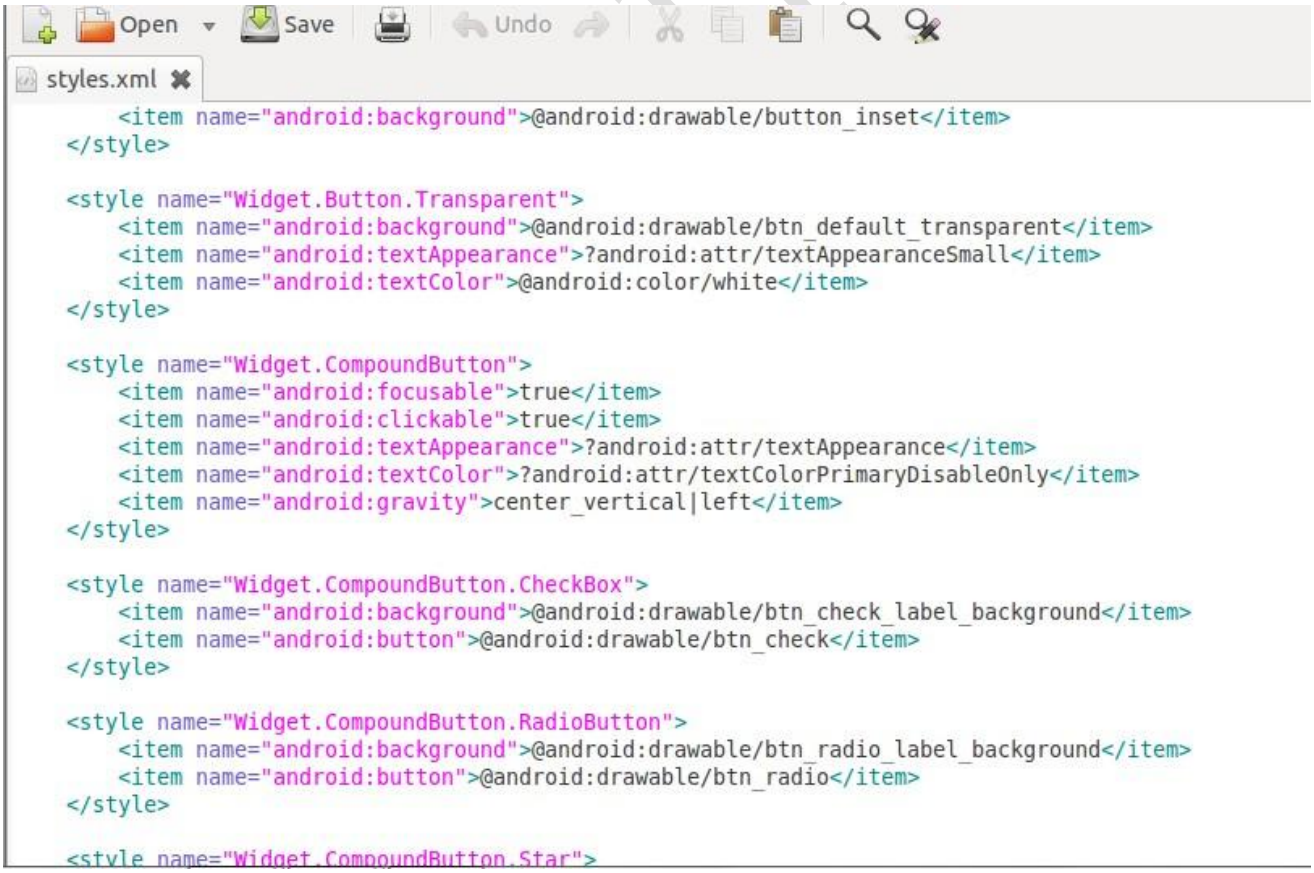
这个方法是最重要的。如果想把控件显示给用户，就必须覆盖次方法。在这个方法主要使用画布，画笔。如果展现给用户的空间要不就是用画笔画出来，要不就是使用图片。当然，两者可以结合使用。实现自定义View的最主要的是重写onDraw(Canvas canvas)函数，当每次系统重绘界面的时候，都会调用这个函数，并传下一个Canvas，在这个函数内，应该将这个View所要显示的内容都draw到这个Canvas上，界面显示出来的内容几乎都由这个Canvas来决定。Canvas的具体画法可以很容易查得到，应该说Android内所有函数的命名都是很直观，一目了然的，自己看一下函数名都大概可以明白这个函数是有什么用的。SDK也是查询Android API的最好的工具，多使用些肯定有好处的。Android

觉得部分还是用图片的。仔细看到里面的构造函数有一个

```
public Button(Context context, AttributeSet attrs) {
        this(context, attrs, com.android.internal.R.attr.buttonStyle);
}
```

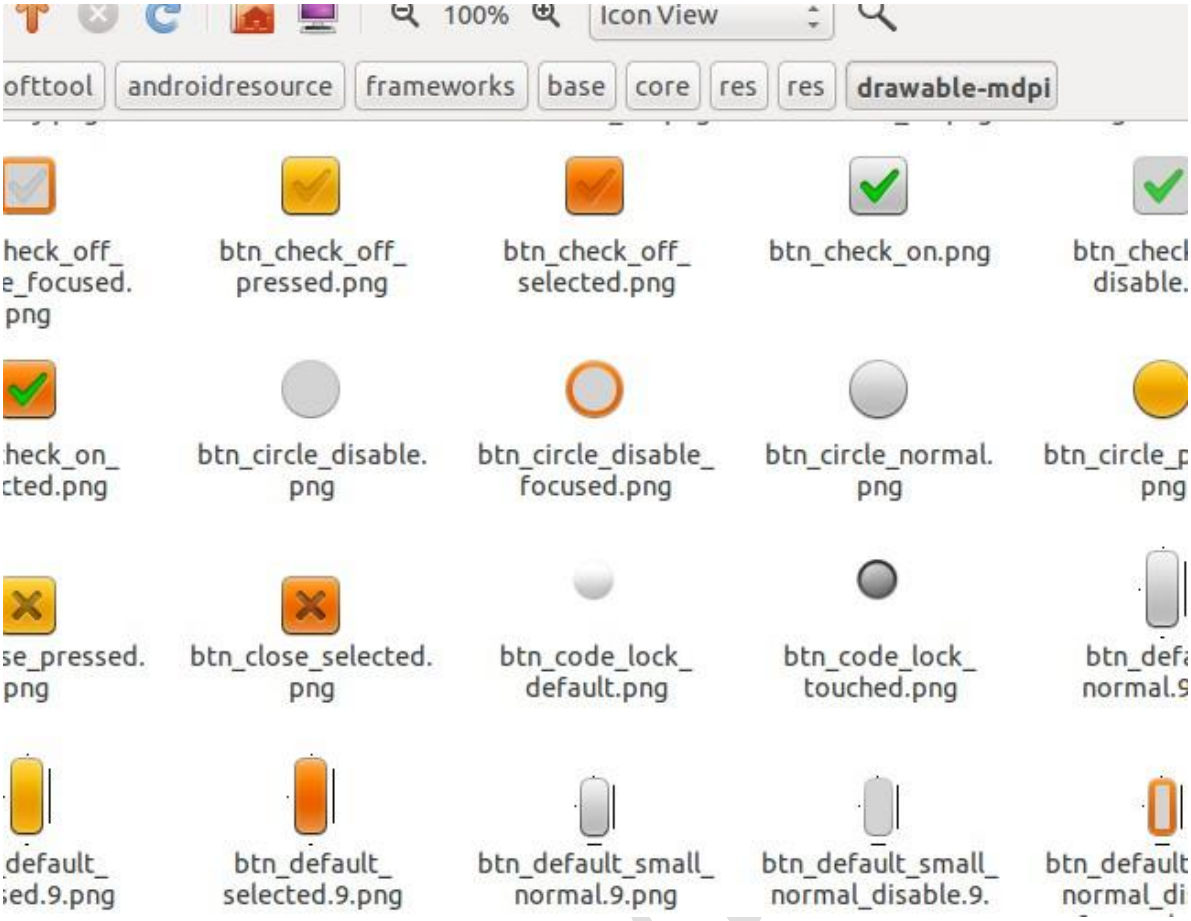
里面就用到了样式





可以看出，在这些配置文件配置了 android 空间的状态。比如，获得焦点的时候切换那张图片，失去焦点的时候切换那张图片，点击的时候切换那张图片等等。可以说 android 自带的控件为什么这么好，也离不开设计这些图片的工程师。我们稍微看看这些系统空

间背后的图片。



其实网上也有很多自定义空间的例子。有些是直接继承现有的 android 控件，有些是 android 控件的父类。但是都是换汤不换药，只要理解 android 系统自带的控件比看什么例子都好。

3.3 三个 Android 自定义控件实例

3.3.1 一个控件,单击的事件不会传给它的 parent

```
public class DontPressWithParentImageView extends ImageView {

    public DontPressWithParentImageView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public void setPressed(boolean pressed) {
        // If the parent is pressed, do not set to pressed.
        if (pressed && ((View) getParent()).isPressed()) {
            return;
        }
        super.setPressed(pressed);
    }
```

```
}
```

3.3.2 可以自由拽托的控件

```
public class SlidingButton extends TextView {

    private float curX;
    private float curY;
    private Paint paint;
    private Bitmap bitmap;
    private boolean isPicked;
    /**
     * 记录点击的位置和上次所在位置的偏移
     */
    private float offSetX = 0;
    /**
     * 记录点击的位置和上次所在位置的偏移
     */
    private float offSetY = 0;

    public SlidingButton(Context context) {
        super(context);
        initLabelView();
    }

    public SlidingButton(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
         initLabelView();
    }

    public SlidingButton(Context context, AttributeSet attrs) {
        super(context, attrs);
         initLabelView();
    }

    private final void initLabelView() {

        paint = new Paint();
        paint.setColor(0xFF888888);
         bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.icon);
     }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
```

```
        switch (event.getAction()) {

        case MotionEvent.ACTION_DOWN:
            if (event.getX()>=curX && event.getX()<=curX+bitmap.getWidth()
                    &&event.getY()>=curY && event.getY()<=curY+bitmap.getHeight()) {

                offSetX = event.getX() - curX;
                offSetY = event.getY() - curY;
                invalidate();
                isPicked = true;
            }
            else
            {
                isPicked = false;
            }

            break;

        case MotionEvent.ACTION_MOVE:

            if (isPicked) {

                curX = event.getX()-offSetX;
                curY = event.getY()-offSetY;
                invalidate();
            }

            break;

        case MotionEvent.ACTION_UP:

            if (isPicked) {

                curX = event.getX()-offSetX;
                curY = event.getY()-offSetY;
                TranslateAnimation tranforAni = new TranslateAnimation(curX,0,curY,0);
                tranforAni.setDuration(1000);
                this.startAnimation(tranforAni);
//              invalidate();
            }
            break;

        default:

            break;
```

```
            }

            return true;
        }

        @Override
        protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

//          super.onMeasure(widthMeasureSpec, heightMeasureSpec);
            MeasureSpec.getSize(widthMeasureSpec);
            MeasureSpec.getSize(heightMeasureSpec);
//          setMeasuredDimension(bitmap.getWidth(), bitmap.getHeight());
            setMeasuredDimension(MeasureSpec.getSize(widthMeasureSpec),
MeasureSpec.getSize(heightMeasureSpec));

        }

        @Override
        protected void onDraw(Canvas canvas) {

//          super.onDraw(canvas);
            canvas.drawBitmap(bitmap, curX, curY, paint);
        }

}
```

3.3.3 launch 界面屏幕切换的控件

```
/**
 * 仿 Launcher 中的 WorkSapce，可以左右滑动切换屏幕的类
 * @author Yao.GUET
 * blog: http://blog.csdn.NET/Yao_GUET
 * date: 2011-05-04
 */
public class ScrollLayout extends ViewGroup {
    private static final String TAG = "ScrollLayout";
    private Scroller mScroller;
    private VelocityTracker mVelocityTracker;

    private int mCurScreen;
    private int mDefaultScreen = 0;

    private static final int TOUCH_STATE_REST = 0;
    private static final int TOUCH_STATE_SCROLLING = 1;
```

```
    private static final int SNAP_VELOCITY = 600;

    private int mTouchState = TOUCH_STATE_REST;
    private int mTouchSlop;
    private float mLastMotionX;
    private float mLastMotionY;
    public ScrollLayout(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
        // TODO Auto-generated constructor stub
    }
    public ScrollLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        // TODO Auto-generated constructor stub
        mScroller = new Scroller(context);

        mCurScreen = mDefaultScreen;
        mTouchSlop =
ViewConfiguration.get(getContext()).getScaledTouchSlop();
    }
    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        // TODO Auto-generated method stub
        if (changed) {
            int childLeft = 0;
            final int childCount = getChildCount();

            for (int i=0; i<childCount; i++) {
                final View childView = getChildAt(i);
                if (childView.getVisibility() != View.GONE) {
                    final int childWidth =
childView.getMeasuredWidth();
                    childView.layout(childLeft, 0,
                            childLeft+childWidth,
childView.getMeasuredHeight());
                    childLeft += childWidth;
                }
            }
        }
    }
    @Override
    protected void onMeasure(int widthMeasureSpec, int
heightMeasureSpec) {
        Log.e(TAG, "onMeasure");
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);

        final int width = MeasureSpec.getSize(widthMeasureSpec);
```

```
            final int widthMode =
MeasureSpec.getMode(widthMeasureSpec);
            if (widthMode != MeasureSpec.EXACTLY) {
                  throw new IllegalStateException("ScrollLayout only
canmCurScreen run at EXACTLY mode!");
            }

            final int heightMode =
MeasureSpec.getMode(heightMeasureSpec);
            if (heightMode != MeasureSpec.EXACTLY) {
                  throw new IllegalStateException("ScrollLayout only can run at
EXACTLY mode!");
            }

            // The children are given the same width and height as the
scrollLayout
            final int count = getChildCount();
            for (int i = 0; i < count; i++) {
                  getChildAt(i).measure(widthMeasureSpec,
heightMeasureSpec);
            }
            // Log.e(TAG, "moving to screen "+mCurScreen);
            scrollTo(mCurScreen * width, 0);
      }

      /**
       * According to the position of current layout
       * scroll to the destination page.
       */
      public void snapToDestination() {
            final int screenWidth = getWidth();
            final int destScreen = (getScrollX()+ screenWidth/2)/screenWidth;
            snapToScreen(destScreen);
      }

      public void snapToScreen(int whichScreen) {
            // get the valid layout page
            whichScreen = Math.max(0, Math.min(whichScreen,
getChildCount()-1));
            if (getScrollX() != (whichScreen*getWidth())) {

                  final int delta = whichScreen*getWidth()-getScrollX();
                  mScroller.startScroll(getScrollX(), 0,
                              delta, 0, Math.abs(delta)*2);
                  mCurScreen = whichScreen;
                  invalidate();           // Redraw the layout
```

```
            }
        }

        public void setToScreen(int whichScreen) {
            whichScreen = Math.max(0, Math.min(whichScreen,
getChildCount()-1));
            mCurScreen = whichScreen;
            scrollTo(whichScreen*getWidth(), 0);
        }

        public int getCurScreen() {
            return mCurScreen;
        }

        @Override
        public void computeScroll() {
            // TODO Auto-generated method stub
            if (mScroller.computeScrollOffset()) {
                scrollTo(mScroller.getCurrX(), mScroller.getCurrY());
                postInvalidate();
            }
        }
        @Override
        public boolean onTouchEvent(MotionEvent event) {
            // TODO Auto-generated method stub

            if (mVelocityTracker == null) {
                mVelocityTracker = VelocityTracker.obtain();
            }
            mVelocityTracker.addMovement(event);

            final int action = event.getAction();
            final float x = event.getX();
            final float y = event.getY();

            switch (action) {
            case MotionEvent.ACTION_DOWN:
                Log.e(TAG, "event down!");
                if (!mScroller.isFinished()){
                    mScroller.abortAnimation();
                }
                mLastMotionX = x;
                break;

            case MotionEvent.ACTION_MOVE:
                int deltaX = (int)(mLastMotionX - x);
```

```
                mLastMotionX = x;

                scrollBy(deltaX, 0);
                break;

        case MotionEvent.ACTION_UP:
                Log.e(TAG, "event : up");
                // if (mTouchState == TOUCH_STATE_SCROLLING) {
                final VelocityTracker velocityTracker = mVelocityTracker;
                velocityTracker.computeCurrentVelocity(1000);
                int velocityX = (int) velocityTracker.getXVelocity();
                Log.e(TAG, "velocityX:"+velocityX);

                if (velocityX > SNAP_VELOCITY && mCurScreen > 0) {
                        // Fling enough to move left
                        Log.e(TAG, "snap left");
                        snapToScreen(mCurScreen - 1);
                } else if (velocityX < -SNAP_VELOCITY
                                && mCurScreen < getChildCount() - 1) {
                        // Fling enough to move right
                        Log.e(TAG, "snap right");
                        snapToScreen(mCurScreen + 1);
                } else {
                        snapToDestination();
                }
                if (mVelocityTracker != null) {
                        mVelocityTracker.recycle();
                        mVelocityTracker = null;
                }
                // }
                mTouchState = TOUCH_STATE_REST;
                break;
        case MotionEvent.ACTION_CANCEL:
                mTouchState = TOUCH_STATE_REST;
                break;
        }

        return true;
}
@Override
public boolean onInterceptTouchEvent(MotionEvent ev) {
        // TODO Auto-generated method stub
        Log.e(TAG, "onInterceptTouchEvent-slop:"+mTouchSlop);

        final int action = ev.getAction();
        if ((action == MotionEvent.ACTION_MOVE) &&
```

```
                    (mTouchState != TOUCH_STATE_REST)) {
                return true;
            }

            final float x = ev.getX();
            final float y = ev.getY();

            switch (action) {
            case MotionEvent.ACTION_MOVE:
                final int xDiff = (int)Math.abs(mLastMotionX-x);
                if (xDiff>mTouchSlop) {
                    mTouchState = TOUCH_STATE_SCROLLING;

                }
                break;

            case MotionEvent.ACTION_DOWN:
                mLastMotionX = x;
                mLastMotionY = y;
                mTouchState = mScroller.isFinished()? TOUCH_STATE_REST :
TOUCH_STATE_SCROLLING;
                break;

            case MotionEvent.ACTION_CANCEL:
            case MotionEvent.ACTION_UP:
                mTouchState = TOUCH_STATE_REST;
                break;
            }

            return mTouchState != TOUCH_STATE_REST;
        }

}
```

【其他】

4.1 提交BUG

如果你发现文档中有不妥的地方，请发邮件至eoeandroid@eoemobile.com 进行反馈，我们会定期更新、发布更新后的版本。

4.2 关于eoeAndroid

eoeAndroid 是国内成立最早，规模最大的Android 开发者社区，拥有海量的Android 学习资料。分享、互助的氛围，让Android 开发者迅速成长，逐步从懵懂到了解，从入门到开发出属于自己的应用，eoeAndroid 为广大Android 开发者奠定坚实的技术基础。从初级到高级，从环境搭建到底层架构，eoeAndroid 社区为开发者精挑细选了丰富的学习资料和实例代码。让Android 开发者在社区中迅速的成长，并在此基础上开发出更多优秀的Android 应用。

北京易联致远无线技术有限公司

责任编辑： QUMIN

美术支持： 金明根　　**技术支持：** haoliuyou

中国最大的Android 开发者社区：www.eoeandroid.com

中国本土的Android 软件下载平台：www.eoemarket.com