预览概述

1. 为什么自定义控件
2. 自定义控件的几种方式
3. onDraw() 画笔 抗锯齿 设置颜色 笔触 叠加模式 xfermode
4. canvas的几种方法 drawPath 旋转、缩放、 保存状态 回复状态钟表控件
5. onMeasure() onLayout()
6. 向上翻滚的文本控件（自定义属性）
7. 可拖动的标签
8. ViewPager 切换效果

自定义控件

1 什么是控件

1. 在屏幕区域展示特定的内容。
2. 能够与用户交互的特殊类。

2 常见的控件

我们常见的控件分成两种，一种是作为容器来放置其他的控件，一种主要的作用是显示内容和交互。

1. LinearLayout、RelativeLayout、FrameLayout...
2. Button、ImageView、TextView、CheckBox...

第一类的父类是ViewGroup,常用来装载各种其他控件。

第二类的父类是View。用于显示或交互的特殊类。

3 为什么要学习自定义控件

1. 为了实现复杂的页面效果。
2. 目前的系统控件无法满足需求，如圆形的ImageView。
3. 了解Android系统控件显示的流程。为进阶准备。

4 自定义控件的方式

1. 组合各种控件,组合各种控件。
2. 修改已有控件，达到需求。
3. 继承View,实现需要的方法，完成自定义控件的设计。

实验：创建一个圆形的按钮

---

1 OnDraw

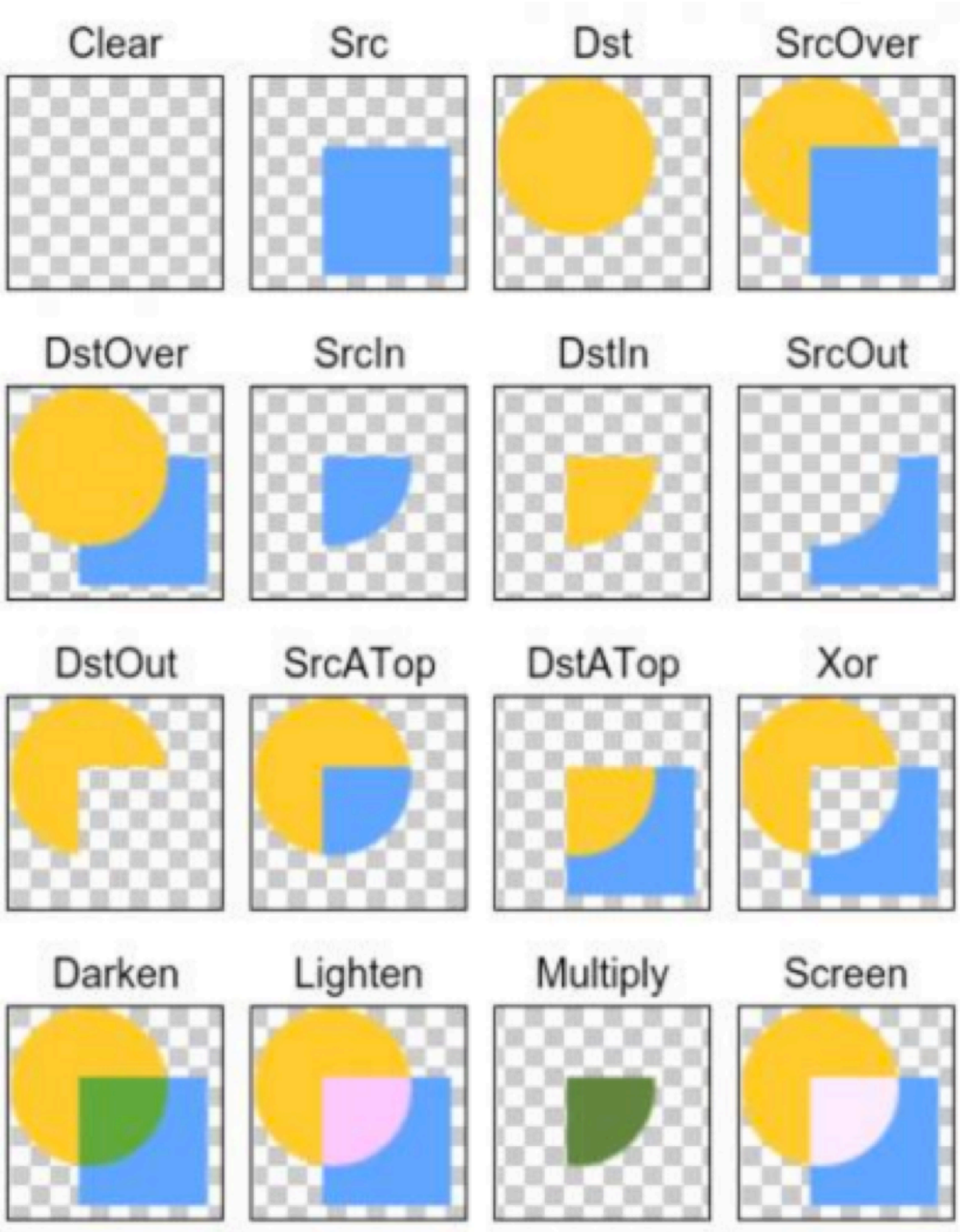
onDraw是View进行显示界面的方法，我们常常在这个方法内部进行绘制控件的外观。涉及到几个比较重要的类:

1. Paint 画笔类，可设置颜色线条宽度等.
2. Canvas 画布类，提供了画各种图形的api
3. Bitmap 位图类，用它来获取图片信息
4. matrix 矩阵类，用它来控制图片的显示

1.1 Paint

Paint 类是一个辅助类，我们常常使用来设置画布的显示效果，如颜色、遮照，图形的线的宽度等等。我们常用的api有:

- setFlags(int flags)设置画笔的一些属性，如抗锯齿
- setColor(int color)设置画笔的颜色
- setStrokeWidth(float width)设置画笔的线宽
- setStyle(Paint.Style style)设置画笔的样式，比如说是否填充图形
- setTextSize(float textSize)设置画文字的大小
- setXfermode(Xfermode xfermode)设置叠加模式



setXfermode 是Paint提供给我们的一个非常重要的api，我们可以使用这个api来开发出酷炫的遮照效果。

上图是叠加效果的示意图，如何看这个图呢？ Src是前景，也就是放在前面的图片，Dst是背景，也就是放在背后的图片。使用setXfermode有两点需要注意：

1. 记得打开硬件加速，否者会出现黑色的内容
2. 画完后记得setXfermode(null);

1.2 Canvas

CanVas画布类，提供画出2D图形以及相关操作的api，我们常使用的三类api和一些辅助类：

第一类，2d图形绘制： - canvas.drawRect 画正方形

- canvas.drawBitmap画图片
- canvas.drawLine 画线
- canvas.drawText 画文字
- canvas.drawOval 画椭圆
- canvas.draArc 画圆弧
- canvas.drawCircle 画圆
- canvas.drawPath 画自定义路径

第二类，画布编辑： - canvas. translate 平移

- canvas.scale 缩放
- canvas.rotate 旋转

第三类，画布状态：

- canvas. Save 保持画布状态
- canvas.restore 恢复画布状态

辅助类：屏幕的一个区域

- Rect(int left, int top, int right, int bottom)
- RectF(float left, float top, float right, float bottom)

初识View的绘制流程

View作为所有控件的父类，View将提供了View显示所有的方法，我们将复写View的方法达到我们需要的功能，View 有点类似 Java 的Object 类，Object 也是提供了很多方法，如equals(Object obj)、toString()

等方法，我们可以重写重写这些方法，达到我们需要实现的功能。

我们可以理解View的绘制过程是一个定制家具的过程，我们可以这么理解如果我们需要定制家具的话，需要这样几个流程：

1. 测量，先测量房间的大小，再根据房间的大小进行计算家具的大小。（onMeasure）
2. 开始定制,根据测量出来的尺寸进行生产。（onDraw）
3. 生产完成，要将定制的家具放置在指定位置。（ViewGroup onLayout）
4. 生产、运输完成后，我们就能使用定制的家具了。（onTouch）

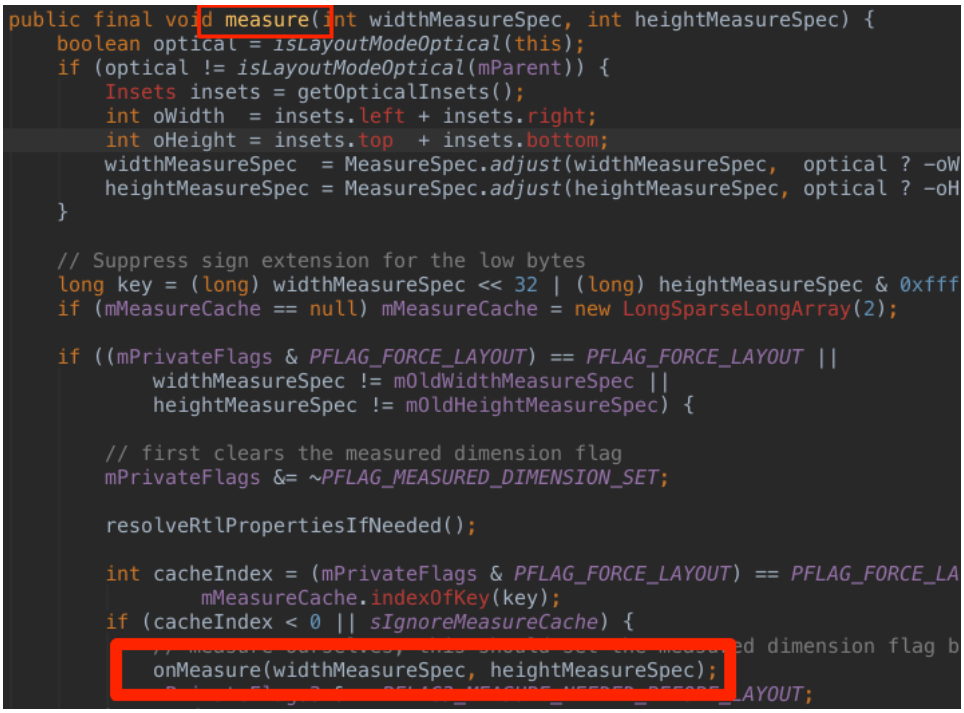
大致的流程是如下: viewGroup会逐一的去调用子控件的测量方法，叫子控件去测量一下自己的宽度与高度，每个控件都会在onMeasure方法完成自己的大小的测量。测量完成后父控件会告诉子控件显示自己的内容出来，而控件的显示内容的方法就在onDraw中实现。最后viewGroup 会在onLayout 中将控件按照一定的规则排列，到现在，控件的内容就展示在我们眼前了，如果我们需要控件能够和我们的手指进行交互，还需要实现onTouch。

2 onMeasure

在自定义控件的时候，我们常常需要确定一个控件的显示边界，onMeasure是android提供给我们的一个计算控件大小的方法，父控件会通过onMeasure将他希望的布局参数传递给当前的View。

2.1 onMeasure的流程

首先根节点的measure方法会被调用，由于measure是view的final方法,子类是无法重写的，measure的方法体内会回调onMeasure，所以实际上测量的工作是在onMeasure内完成的，而且由此我们可以知道measure是提供给外部调用的一个测量自己的方法。由于根节点大部分是ViewGroup,如LinerLayout等，所以ViewGroup会在onMeasure方法内部逐一的调用子控件的measure方法，一直到最后的一个子控件。实际上是一个自顶向下的过程。



2.2 LinearLayout onMeasure 流程分析

根据上面的流程我们知道了，onMeasure是完成控件大小测量的实际操作的步骤，下面我们来分析一下平常使用最多的LinearLayout是如何完成onMeasure的。

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    if (mOrientation == VERTICAL) {
     measureVertical(widthMeasureSpec, heightMeasureSpec);
    } else {
     measureHorizontal(widthMeasureSpec, heightMeasureSpec);
    }
}
```

LinearLayout根据设置水平对齐还是垂直对齐分别不同的计算方式 measureVertical 计算垂直高度，measureHorizontal 计算水平宽度。下面我们以measureVertical为例子进行分析

```
void measureVertical(int widthMeasureSpec, int heightMeasureSpec) {
    mTotalLength = 0;
    int maxWidth = 0;
    int childState = 0;
    int alternativeMaxWidth = 0;
    int weightedMaxWidth = 0;
    boolean allFillParent = true;
    float totalWeight = 0;
    //获取所有子控件的个数
    final int count = getVirtualChildCount();
    //获取宽度布局的模式
     final int widthMode =   MeasureSpec.getMode(widthMeasureSpec);
     //获取高度布局的模式
    final int heightMode = MeasureSpec.getMode(heightMeasureS);
          .
          .
          .
 //开始遍历每一个子控件
for (int i = 0; i < count; ++i) {
   final View child = getVirtualChildAt(i);

        //如果子控件为空，到下一个子控件
        if (child == null) {
            mTotalLength += measureNullChild(i);
            continue;
        }
        //如果子控件为不可见，直接到下一个子控件
        if (child.getVisibility() == View.GONE) {
           i += getChildrenSkipCount(child, i);
           continue;
        }

        //获取到子控件的布局参数
         LinearLayout.LayoutParams lp = (LinearLayout.LayoutParams) child.getLayoutParams();
          .
          .
          .
       //测量子控件的核心方法
      measureChildBeforeLayout(
           child, i, widthMeasureSpec, 0, heightMeasureSpec,
                   totalWeight == 0 ? mTotalLength : 0);
          .
          .
          .
    }
```

看完measureVertical方法后，我们发现实际上进行测量的方法是measureChildBeforeLayout，我们再看看这个的方法实现。

```
 void measureChildBeforeLayout(View child, int childIndex,int widthMeasureSpec, int totalWidth, int heightMeasureSpec,int
totalHeight) {

       //测量子控件的核心方法
    measureChildWithMargins(child, widthMeasureSpec, totalWidth,heightMeasureSpec, totalHeight);

}
```

看完measureChildBeforeLayout方法后，我们发现实际上进行测量的方法是measureChildWithMargins，这个是一个ViewGroup提供的方法，也就是说，每一个viewGroup的继承类都能使用这个方法。接着我们继续往下看。

```
 protected void measureChildWithMargins(View child,
        int parentWidthMeasureSpec, int widthUsed,
        int parentHeightMeasureSpec, int heightUsed)    {
    final MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();

    final int childWidthMeasureSpec = getChildMeasureSpec(parentWidthMeasureSpec,
            mPaddingLeft + mPaddingRight + lp.leftMargin + lp.rightMargin
                    + widthUsed, lp.width);


    final int childHeightMeasureSpec = getChildMeasureSpec(parentHeightMeasureSpec,
            mPaddingTop + mPaddingBottom + lp.topMargin + lp.bottomMargin
                    + heightUsed, lp.height);

   //实际起左右的语句就是调用了View.measure方法
   child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
}
```

最后我们发现在measureChildWithMargins，实际上还是调用了child.measure(childWidthMeasureSpec, childHeightMeasureSpec);所以也就验证了我们关于view的测量的流程，控件从ViewGroup开始，逐一的找到它的子控件，根据子控件的布局参数，进行封装，最后调用measure，然后在子控件的onMeasure方法完成测量。

2.3 onMeasure(int widthMeasureSpec, int heightMeasureSpec)

既然刚刚我们根据查看源码发现，控件的大小是需要在onMeasure方法内部完成测量，那widthMeasureSpec与heightMeasureSpec这两个形参是什么呢？我们在layout的布局文件内部的定义的layout_width与layout_height，与这两个参数有什么关联呢？ 在进行测量之前我们要弄清楚几个问题：

1. 子控件显示在父控件的内部，大部分子控件的宽高被父控件的宽高限制。
2. 如ScrollView等控件，由于能够滑动，就不限制子控件的高度。

实际上layout_width，layout_height是控件向父控件申请的空间。我们常常有以下几种值可以选择：

- match_parent
- wrap_content
- xxxdpi

这三种模式，我们可以分成两种模式，第一种是固定大小，match_parent，xxxdpi，match_parent是指与父控件一样大。另外一种是不固定大小的，wrap_content，根据内容的大小进行显示。

我们设置在xml上面时，控件就能够根据这个设置进行更改大小，控件到底是如何获取到这个值得呢？

实际上控件是通过widthMeasureSpec和heightMeasureSpec获取到layout_width，layout_height，widthMeasureSpec和heightMeasureSpec是对layout_width进行了一次包装，这个int值里面包含了两个值，一个是mode（模式），一个是size（大小）。

我们可以使用以下方法获取到widthMeasureSpec和heightMeasureSpec的值

- MeasureSpec.getMode(measureSpec);//得到模式
- MeasureSpec.getSize(measureSpec);//得到大小

MeasureSpec提供了三种类型：

1. MeasureSpec.AT_MOST
2. MeasureSpec.EXACTLY
3. MeasureSpec.UNSPECIFIED

这三种模式分别对应： MeasureSpec.AT_MOST--wrap_content，由控件内容来决定大小，但是最大不能超过父控件的大小。

MeasureSpec.EXACTLY--match_parent,xxxdpi,已经指定了特定的大小。match_parent 与父控件一样大，所以也是已经知道指定的大小

MeasureSpec.UNSPECIFIED，对控件的大小不做限制，控件可以要多大都行，比如说ScrollView的子控件。

最后根据widthMeasureSpec 和 heightMeasureSpec 计算出需要的大小后，需调用setMeasuredDimension才能起效。

2.4 View 常见的 onMeasuere 写法

```
private int resolveAdjustedSize(int desiredSize, int maxSize,  int measureSpec) {
int result = desiredSize;
int specMode = MeasureSpec.getMode(measureSpec);
  int specSize =  MeasureSpec.getSize(measureSpec);
      switch (specMode) {

         case MeasureSpec.UNSPECIFIED:

         result = Math.min(desiredSize, maxSize);

         break;

         case MeasureSpec.AT_MOST:

         result = Math.min(Math.min(desiredSize, specSize), maxSize);

         break;

         case MeasureSpec.EXACTLY:

         result = specSize;

         break;
                  }
       return result;
}
```

desiredSize 是需要使用的尺寸，maxSize 是最大使用的尺寸，measureSpec 是onMeasure传进来的参数。

- UNSPECIFIED，无限制尺寸，只要能满足控件的需求即可，所以比较 desiredSize 和 maxSize 的大小，去最小的即可
- AT_MOST，wrapContent,也就是说这个控件最大不能超过父控件的大小，所以先比较desiredSize和specSize的大小，去最小的，再和maxSize比较。

- EXACTLY，指定尺寸的，直接返回specSize即可。

2.5 ViewGroup 常见的 onMeasuere 写法

viewGroup有义务对子控件的大小进行测量，ViewGroup提供了一下几个方法来调用子控件的自我测量

1. measureChildren(int widthMeasureSpec, int heightMeasureSpec);
2. measureChild(View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec);
3. measureChildWithMargins(View child, int parentWidthMeasureSpec, int widthUsed, int parentHeightMeasureSpec, int heightUsed)

ViewGroup 的 onMeasuere 常常需要和onLayout一起使用，原因是在放置子控件的时候，我们常常是需要知道子控件的大小，才能够放到合适的位置上。

3 onLayout

该方法是提供给ViewGroup来放置每个子控件的。通常我们需要配onMeasure来一起使用。

这个方法是每次布局大小变化，或者位置改变的时候都会本调用。

onLayout(boolean changed, int l, int t, int r, int b) changed 这个控件是否更改位置或者尺寸了

l 相对于父控件的左边的位置

t 相对于父控件的顶部的位置

r 相对于父控件的右边的位置

b 相对于父控件的下部的位置

我们必须在这个方法内部将子控件有序的放置在合适的位置。View 提供了llayout(int l, int t, int r, int b)方法给我们。

综合练习 01：向上翻滚的文本

- 新建一个类继承ViewFlipper
- 重写这个类的构造方法
- 编写文字进入和退出的动画 进入动画，从底部进入，并带有渐变的效果

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="500"
        android:fromYDelta="100%p"
        android:toYDelta="0"/>
    <alpha
        android:duration="500"
        android:fromAlpha="0.0"
        android:toAlpha="1.0"/>
</set>
```

上移动画

显示动画

退出动画

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="500"
        android:fromYDelta="0"
        android:toYDelta="-100%p"/>
    <alpha
        android:duration="500"
        android:fromAlpha="1.0"
        android:toAlpha="0.0"/>
</set>
```

划出控件顶部

- 在构造方法内部初始化viewFlipper的参数

```
public void init(Context mContext){
    setFlipInterval(2000);

    Animation animIn = AnimationUtils.loadAnimation(mContext, R.anim.anim_marquee_in);
    animIn.setDuration(200);
    setInAnimation(animIn);

    Animation animOut = AnimationUtils.loadAnimation(mContext, R.anim.anim_marquee_out)
    animOut.setDuration(200);
    setOutAnimation(animOut);
}
```

setFlipInterval(); 设置翻牌的间隔 加载翻牌显示的动画 setInAnimation(); 设置进入的动画 setOutAnimation(); 设置出去的动画

- 编写创建各个子TextView的方法

```
// 创建ViewFlipper下的TextView
private TextView createTextView(String text, int position) {
    TextView tv = new TextView(mContext);
    tv.setGravity(Gravity.CENTER);
    tv.setText(text);
    tv.setTextColor(Color.BLACK);
    tv.setTextSize(23);
    tv.setTag(position);
    return tv;
}
```

- 往TextFlipper内部添加数据

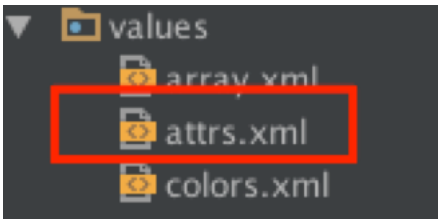
```
// 启动轮播
public boolean start() {
    if (notices == null || notices.size() == 0) return false;
    removeAllViews();

    for (int i = 0; i < notices.size(); i++) {
        final TextView textView = createTextView(notices.get(i), i);
        final int finalI = i;
        textView.setOnClickListener((v) -> {
                if (onItemClickListener != null) {
                    onItemClickListener.onItemClick(finalI, textView);
                }
        });
        addView(textView);
    }

    if (notices.size() > 1) {
        startFlipping();
    }
    return true;
}
```

removeAllViews() 首先删除控件的子控件

addView() 接着把需要添加的TextView 逐一添加到控件中

startflipping() 开始滚动

- 定义自定义属性
  在values文件夹下创建attrs.xml



- 在attrs.xml文件内部添加需要添加的属性

```
<declare-styleable name="TextFlipper">    属性集合
    <attr name="textColor" format="color"/>
    <attr name="textSize" format="dimension"/>
某一属性<attr name="texts" format="reference"/>
</declare-styleable>
```

declare-styleable name 定义的是属性集合的名称

attr name 定义的是某一个属性的名称

- 使用属性 如果我们需要使用自动属性，我们需自定义命名空间

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

红框的是命名空间的名称

```
<com.it520.textfliper.TextFlipper
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:id="@+id/flipper"
    app:textColor="@color/colorPrimary"
    app:textSize="10sp"
    app:texts="@array/title"
    />
```

使用自定义的属性

命名空间：属性名称= "format 类型"

- 接收属性值

```
TypedArray array =  context.obtainStyledAttributes(a
mTextColor =  array.getColor(R.styleable.TextFlippe
mTextSize = array.getDimensionPixelSize(R.styleable.
int titles  = array.getResourceId(R.styleable.TextF
```

接收属性使用TypedArry类进行获取

自定义控件的属性

我们在使用系统控件的时候，常常能够在xml上面进行配置，提供了这类属性的控件能够更加便于开发者的使用和维护，所以我们学习如何定义控件的属性非常有必要。 下面就是自定义控件属性的步骤：

1. 在资源文件夹下新建文件attrs.xml
2. 在attrs.xml里面新建需要定义的属性
3. 新建一个declare-styleable 为这个属性增加一个name属性如 declare-styleable name="---"
4. 在declare-styleable的内部定义需要的属性，如 ， name是属性的名称，format是属性的类型

5. 在自定义控件接收自定义属性，在构造函数中使用Context获取TypedArray，obtainStyledAttributes(attrs, R.styleable.定义的属性)
6. 使用TypedArray.getxxx方法来获取相应的值，这个xxx就是定义在attr里面的类型，R.styleable.（declare-styleable name)_(attr name)
7. 在Xml使用
8. 定义自定义的命名空间，不同的IDE有不一样的处理方法，Eclipse xmlns:自定义名称 ="http://schemas.android.com/apk/res/包名" android studio: xmlns:自定义名称 ="http://schemas.android.com/apk/res-auto"

需要注意的是：

- format 的类型说明见下表
- 在自定义控件中获取属性，需使用类似格式R.styleable.（declare-styleable name)_(attr name)

format的类型

1. reference：参考某一资源ID。
2. color：颜色值
3. boolean：布尔值
4. dimension：尺寸值
5. float：浮点值。
6. integer：整型值。
7. string：字符串
8. fraction：百分数。
9. enum：枚举值
10. flag：位或运算

综合练习 02：可拖动的标签

- 新建一个项目
- 新建一个类继承于Viewgroup

```
<com.m520it.myslideflag.MySlideFlag
    android:id="@+id/myslideflag"
    android:layout_width="wrap_content"
    android:layout_height="match_parent" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/tabicon" />
</com.m520it.myslideflag.MySlideFlag>
```

- 报错 重写onLayout方法
- 重写两个构造方法
- 重写onMeasure，控件的宽度就是拖动按钮的宽度，高度就是父控件的高度

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    //获取子控件 该容器也只有一个子控件
    mChildView = (ImageView) getChildAt(0);
    //测量子控件宽高
    measureChild(mChildView, widthMeasureSpec, heightMeasureSpec);
    mImageWidht = mChildView.getMeasuredWidth();
    mImageHeight = mChildView.getMeasuredHeight();

    mScreenHeight = MeasureSpec.getSize(heightMeasureSpec);
    //设置整个控件宽度为图片宽度 高度为全屏
    setMeasuredDimension(mImageWidht, mScreenHeight);
}

@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    //将子控件布局在左上角
    mChildView.layout(0, 0, mImageWidht, mImageHeight+mTop);
}
```

- 重写onTouch方法
- 当手指在控件内部滑动时，记下手指的位置，减去按钮高度的一半，刷新控件

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:

            break;
        case MotionEvent.ACTION_MOVE:
            int y=(int) event.getY();
            mTop=y-mImageHeight/2;
            break;
        case MotionEvent.ACTION_UP:

            break;
    }
    //重排布子控件
    requestLayout();
    return true;
}

@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
        mChildView.layout(0, mTop, mImageWidht, mImageHeight+mTop);
}
```

- 在onlayout方法内部进行控制，做好子控件的越界处理

```
@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    if (mTop<0) {
        mChildView.layout(0, 0, mImageWidht, mImageHeight);
    }else if (mTop>mScreenHeight-mImageHeight) {
        mChildView.layout(0, mScreenHeight-mImageHeight, mImageWidht, mScreenHeight);
    }else {
        mChildView.layout(0, mTop, mImageWidht, mImageHeight+mTop);
    }
}
```

- 为了优化体验，只有一开始点击了按钮才能拖动

  计算按下的坐标点，如果按下的坐标在拖动块内部，接收后面的事件

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            RectF rect=new RectF(0,mTop,mImageWidht,mTop+mImageHeight);
            float touchX=event.getX();
            float touchY=event.getY();
            if (!rect.contains(touchX, touchY)) {
                //设置返回false 这样就不能接收到其他事件了
                return false;
            }
            mChildView.setImageResource(R.drawable.tabicon_p);

            break;
        case MotionEvent.ACTION_MOVE:
            ..
            break;
        case MotionEvent.ACTION_UP:
            mChildView.setImageResource(R.drawable.tabicon);

            break;
    }
    //重排布子控件
    requestLayout();
    return true;
}
```

- 为控件添加点击事件，当点击的时间小于500毫秒时，我们认为就是点击事件。

```
private SlideFlagClickListener mListener;

public void setOnSlideFlagClickListener(SlideFlagClickListener listener) {
    this.mListener=listener;
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            ..
            //控制点击事件
            mDownTime = System.currentTimeMillis();
            break;
        case MotionEvent.ACTION_MOVE:
            ..
            break;
        case MotionEvent.ACTION_UP:
            ..
            long upTime=System.currentTimeMillis();
            if (upTime-mDownTime<500) {
                if (mListener!=null) {
                    mListener.onSlideFlagClick();
                }
            }
            break;
    }
    //重排布子控件
    requestLayout();
    return true;
}
```

ViewPager创建具有切换效果的页面容器

1 Android support Library 介绍

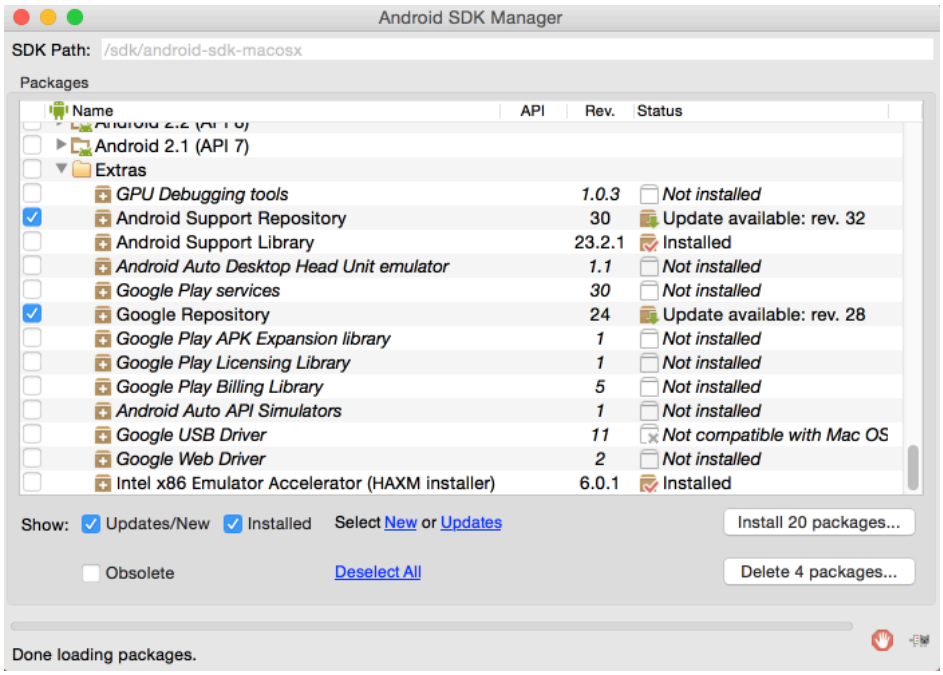
因为Android版本迭代很快，从原来1.3~6.0，Google在高版本出现了很多的Api,如果我们想在低版本使用的话，可能就需要使用Google提供的Support Library。

每种Support Library都提供了不一样的功能，下面我们来简介一下我们常用的Support Library。

- v4 Support Library（为 android 1.6 - api 4 以及以上版本提供内容的拓展包，里面有 Fragment、NotificationCompat、ViewPager、AsyncTaskLoader等内容）
- v7 Support Library(为 android 2.1 - api 7 以及以上的版本提供内容的扩展包，里面有ActionBar、cardview、recyclerview等内容)
- v8 Support Library(为 android 2.3 - api 8 以及以上的版本提供了毛玻璃的效果)
- v13 Support Library
- v14 Support Library
- Multidex Support Library(为 方法数超过65535 的应用提供解决方案的包)

如何使用Support Library

1. 使用Gradle依赖下载 https://developer.android.com/topic/libraries/support-library/features.html#multidex
2. 使用SDK Manager下载到本地（Eclipse 已经停止更新）



2 ViewPager的使用

ViewPager是谷歌提供给我们进行滑动切换页面的一个工具类，如果我们需要使用ViewPager的话，最少涉及到2个类

1. ViewPager
2. PagerAdapter - ViewPager的适配器，为viewpager 提供内容

3 PagerAdapter

如果我们需要使用PagerAdapter为ViewPager提供数据，我们必须实现下面几个方法：

1. instantiateItem(ViewGroup, int)
2. destroyItem(ViewGroup, int, Object)
3. getCount()
4. isViewFromObject(View, Object)

instantiateItem(ViewGroup container, int position) 生成对应位置的页面，container 就是显示页面的容器，position 就是对应的页面的序号。这个方法有一个返回值，返回值的类型是Objcet,这个Object 对象就是对应页面。

destroyItem(ViewGroup, int, Object)将指定页面删除，container 就是显示页面的容器，position 就是对应的位置。

getCount 需要返回一个int值，这个int值就是viewPage需要显示的页面个数。

isViewFromObject(View, Object)，view 是我们某个位置的页面，Object是 instantiateItem 方法返回的。我们在这个方法需要判断这页面与Object对象是否是同一个对象。

4 PageTransformer

ViewPager有个方法叫做： setPageTransformer(boolean reverseDrawingOrder, PageTransformer transformer) 用于设置ViewPager切换时的动画效果，并且google官方还给出了两个示例。 只需要在上述代码中调用setPageTransformer即可添加切换动画效果~~下面演示google的两个PageTransformer的代码，以及运行效果。

1、DepthPageTransformer

```
public class DepthPageTransformer implements ViewPager.PageTransformer {
    private static final float MIN_SCALE = 0.75f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();

        if (position < -1) { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 0) { // [-1,0]
            // Use the default slide transition when moving to the left page
            view.setAlpha(1);
            view.setTranslationX(0);
            view.setScaleX(1);
            view.setScaleY(1);

        } else if (position <= 1) { // (0,1]
            // Fade the page out.
            view.setAlpha(1 - position);

            // Counteract the default slide transition
            view.setTranslationX(pageWidth * -position);

            // Scale the page down (between MIN_SCALE and 1)
            float scaleFactor = MIN_SCALE
                    + (1 - MIN_SCALE) * (1 - Math.abs(position));
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

        } else { // (1,+Infinity]
            // This page is way off-screen to the right.
            view.setAlpha(0);
        }
    }
}
```

调用代码:

```
mViewPager.setPageTransformer(true, new DepthPageTransformer());
```

2、ZoomOutPageTransformer

```
import android.annotation.SuppressLint;
import android.support.v4.view.ViewPager;
import android.util.Log;
import android.view.View;

public class ZoomOutPageTransformer implements ViewPager.PageTransformer
{
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.5f;

    @SuppressLint("NewApi")
    public void transformPage(View view, float position)
    {
        int pageWidth = view.getWidth();
        int pageHeight = view.getHeight();

        Log.e("TAG", view + " , " + position + "");

        if (position < -1)
        { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 1) //a页滑动至b页 ； a页从 0.0 -1 ；b页从1 ~ 0.0
        { // [-1,1]
            // Modify the default slide transition to shrink the page as well
            float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(position));
            float vertMargin = pageHeight * (1 - scaleFactor) / 2;
            float horzMargin = pageWidth * (1 - scaleFactor) / 2;
            if (position < 0)
            {
                view.setTranslationX(horzMargin - vertMargin / 2);
            } else
            {
                view.setTranslationX(-horzMargin + vertMargin / 2);
            }

            // Scale the page down (between MIN_SCALE and 1)
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

            // Fade the page relative to its size.
            view.setAlpha(MIN_ALPHA + (scaleFactor - MIN_SCALE)
                    / (1 - MIN_SCALE) * (1 - MIN_ALPHA));

        } else
        { // (1,+Infinity]
            // This page is way off-screen to the right.
            view.setAlpha(0);
        }
    }
}
```

以下是具体切换页面效果的资料

http://blog.csdn.net/lmj623565791/article/details/40411921/

5 OnPageChangeListener ViewPager滚动监听器

当我们需要监听ViewPager滚动状态的时候，我们需要为ViewPager设置一个滚动监听器。

当ViewPager滚动的时候，会调用一下个方法：

1. onPageScrollStateChanged(int state)
2. onPageScrolled(int position, float positionOffset, int positionOffsetPixels)
3. onPageSelected(int position)

onPageScrollStateChanged(int state)，当viewPager滚动状态改变时被调用，state 为当前viewPager的状态。

SCROLL_STATE_IDLE = 0 （滚动停止）

SCROLL_STATE_SETTLING = 2 (手放开，自动滚动)

SCROLL_STATE_DRAGGING = 1 (拖动Viewpager)

(int position) viewPager滚动到某一页，position为滚动到的页数。
