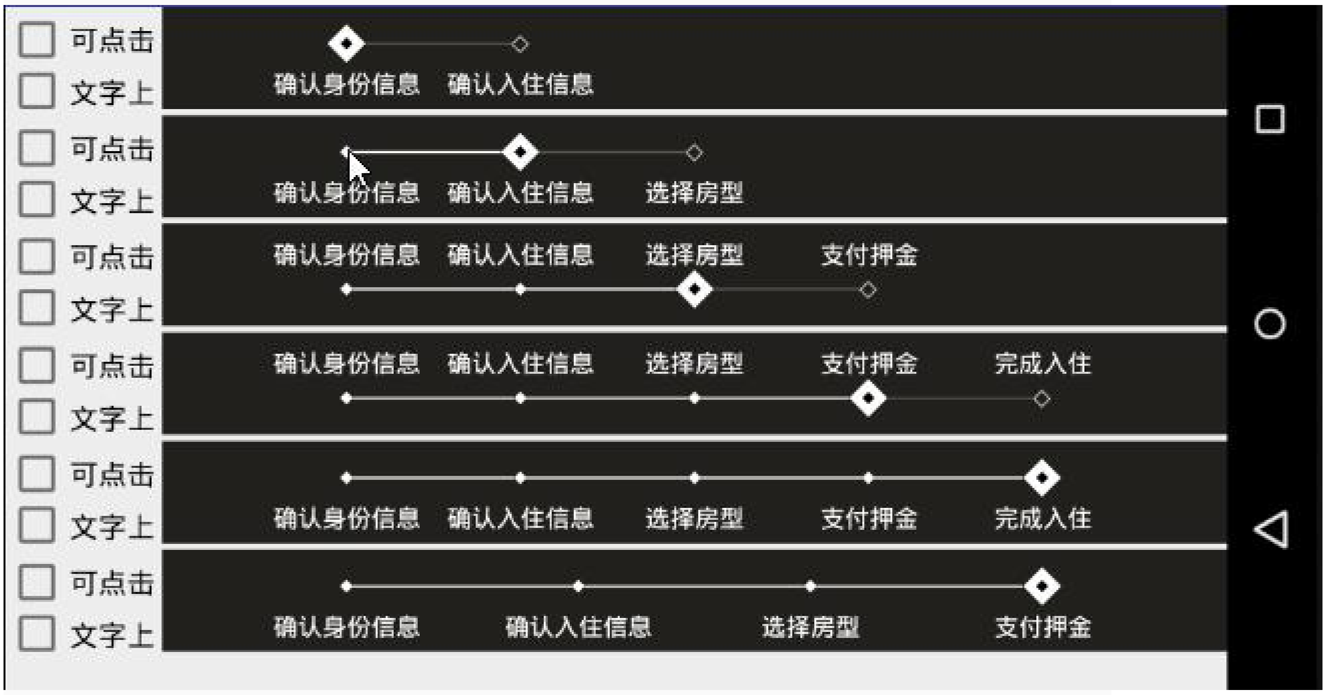
-

自定义 View 的流程和步骤

1.效果图

好了，扯了上面的这些闲话，直接来看效果图吧。



2.实现思路

-

  首先是画各步骤点之间的线条

-
-

  接着是画未选中步骤点的图标

-
-

  第三步是画选中步骤点的图标

-
-

  最后画出各步骤点对应的说明文字

-

3.实现细节

3.1 概述

StepView 继承自 View，通过构造方法初始化一些必要参数，然后在 onSizeChanged 方法中获取 View 的宽高以及其他额外计算的数据信息，最后通过 onDraw 方法绘制出 View。

3.2 首先通过 res/values/attrs 定义一些细节参数

```xml
<declare-styleable name="StepView">

        <!--步骤点个数-->

        <attr name="count" format="integer" />

        <!--链接步骤点之间的线条颜色以及线宽-->

        <attr name="normal_line_color" format="color" />

        <attr name="passed_line_color" format="color" />

        <attr name="line_stroke_width" format="dimension" />

        <!--文字说明信息-->

        <attr name="text_color" format="color" />

        <attr name="text_size" format="dimension" />

        <attr name="text_to_line_margin" format="dimension" />

        <!--边距-->

        <attr name="margin" format="dimension" />

        <!--默认文字在线条下面，线条距顶部距离、文字距底部距离-->

        <attr name="line_to_top_margin" format="dimension" />

        <attr name="text_to_bottom_margin" format="dimension" />

        <!--设置步骤点在哪儿-->

        <attr name="step" format="enum">

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

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

            <enum name="three" value="2" />

            <enum name="four" value="3" />

            <enum name="five" value="4" />
        </attr>

        <!--线条长度是否可变,默认是-->

        <attr name="line_length" format="enum">

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

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

        </attr>

        <!--根据最大步骤点数量，计算出线条长度不变时线条长度，线条长度可变时，该数据无效-->

        <attr name="max_dot_count" format="integer" />

        <!--文字是否在线条下面，默认是-->

        <attr name="text_location" format="enum">

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

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

        </attr>

        <!--此 view 是否可点击-->

        <attr name="is_view_clickable" format="boolean" />

    </declare-styleable>
```

3.3 通过构造方法初始化

```
public StepView(Context context) {

        this(context, null);

    }



    public StepView(Context context, @Nullable AttributeSet attrs) {

        this(context, attrs, 0);

    }



    public StepView(Context context, @Nullable AttributeSet attrs, int defStyleAttr) {

        super(context, attrs, defStyleAttr);

        init(context, attrs, defStyleAttr);
```

```java
    }

private void init(Context context, AttributeSet attrs, int defStyleAttr) {

        defaultNormalLineColor = Color.parseColor("#545454");

        defaultPassLineColor = Color.WHITE;

        defaultTextColor = Color.WHITE;

        defaultLineStikeWidth = dp2px(context, 1);

        defaultTextSize = sp2px(context, 80);

        defaultText2DotMargin = dp2px(context, 15);

        defalutMargin = dp2px(context, 100);

        defaultLine2TopMargin = dp2px(context, 30);

        defaultText2BottomMargin = dp2px(context, 20);



        normal_pic = BitmapFactory.decodeResource(getResources(), R.drawable.ic_normal);

        target_pic = BitmapFactory.decodeResource(getResources(), R.drawable.ic_target);

        passed_pic = BitmapFactory.decodeResource(getResources(), R.drawable.ic_passed);



        TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.StepView,
defStyleAttr, 0);

        dotCount = a.getInt(R.styleable.StepView_count, defaultDotCount);

        if (dotCount < 2) {

            throw new IllegalArgumentException("Steps can't be less than 2");

        }

        stepNum = a.getInt(R.styleable.StepView_step, defaultStepNum);

        lineLength = a.getInt(R.styleable.StepView_line_length, defaultLineLength);

        maxDotCount = a.getInt(R.styleable.StepView_max_dot_count, defaultMaxDotCount);

        if (maxDotCount < dotCount) {//当最多点小于设置点数量时，设置线条长度可变

            lineLength = defaultLineLength;

        }
        textLocation = a.getInt(R.styleable.StepView_text_location, defaultTextLocation);

        isTextBelowLine = textLocation == defaultTextLocation;



        normalLineColor = a.getColor(R.styleable.StepView_normal_line_color,
defaultNormalLineColor);

        passLineColor = a.getColor(R.styleable.StepView_passed_line_color,
defaultPassLineColor);

        lineStikeWidth = a.getDimension(R.styleable.StepView_line_stroke_width,
defaultLineStikeWidth);

        textColor = a.getColor(R.styleable.StepView_text_color, defaultTextColor);

        textSize = a.getDimension(R.styleable.StepView_text_size, defaultTextSize);

        text2LineMargin = a.getDimension(R.styleable.StepView_text_to_line_margin,
defaultText2DotMargin);

        margin = (int) a.getDimension(R.styleable.StepView_margin, defalutMargin);

        line2TopMargin = a.getDimension(R.styleable.StepView_line_to_top_margin,
defaultLine2TopMargin);

        text2BottomMargin = a.getDimension(R.styleable.StepView_text_to_bottom_margin,
defaultText2BottomMargin);

        clickable = a.getBoolean(R.styleable.StepView_is_view_clickable,
defaultViewClickable);

        a.recycle();

        //当文字在线条上面时，参数倒置

        if (!isTextBelowLine) {

            line2BottomMargin = line2TopMargin;

            text2TopMargin = text2BottomMargin;

        }

        //线条画笔

        linePaint = new Paint();

        linePaint.setAntiAlias(true);

        linePaint.setStrokeWidth(lineStikeWidth);
        //文字画笔

        textPaint = new Paint();

        textPaint.setAntiAlias(true);

        textPaint.setColor(textColor);

        textPaint.setTextSize(textSize);

        //存放说明文字的矩形

        bounds = new Rect();

    }
```

由这段代码可知，通过 init 方法，不但初始化了上面的细节参数，还额外初始化了 bitmap、paint、bounds 等参数。由于调用了 dp/sp2px 方法，所以先贴一下该方法。

```
private int dp2px(Context context, int value) {

        float density = context.getResources().getDisplayMetrics().density;

        return (int) (density * value + 0.5f);

    }



    private int sp2px(Context context, int value) {

        float scaledDensity = context.getResources().getDisplayMetrics().scaledDensity;

        return (int) (value / scaledDensity + 0.5f);

    }
```

3.4 在 onSizeChanged 方法中，完成宽高等数据计算。

```
    @Override

    protected void onSizeChanged(int w, int h, int oldw, int oldh) {

        width = w - margin * 2;

        height = h;

        //线条长度是否可变

        if (lineLength == defaultLineLength) {//可变

            perLineLength = width / (dotCount - 1);

        } else {//固定
```

```
            perLineLength = width / (maxDotCount - 1);

        }

        passWH = calculateWidthAndHeight(passed_pic);

        normalWH = calculateWidthAndHeight(normal_pic);

        targetWH = calculateWidthAndHeight(target_pic);

    }
```

此处说明一下，计算 bitma 宽高的方法，然后把宽高信息存于二维数组中

```
    /*计算 bitmap 宽高*/

    private int[] calculateWidthAndHeight(Bitmap bitmap) {

        int[] wh = new int[2];

        int width = bitmap.getWidth();

        int height = bitmap.getHeight();

        wh[0] = width;

        wh[1] = height;

        return wh;

    }
```

3.5 通过 onDraw 方法绘制 View

3.5.1 画步骤点之间连线

```
    /*绘制链接步骤点之间的线条*/

    private void drawConnectLine(Canvas canvas, int stepNum) {

        float startX = 0;

        float stopX = 0;

        for (int i = 0; i < dotCount - 1; i++) {

            /*设置线条起点 X 轴坐标*/

            if (i == stepNum) {

                startX = margin + perLineLength * i + targetWH[0] / 2;

            } else if (i > stepNum) {
                startX = margin + perLineLength * i + normalWH[0] / 2;

            } else {

                startX = margin + perLineLength * i + passWH[0] / 2;

            }

            /*设置线条终点 X 轴坐标*/

            if (i + 1 == stepNum) {

                stopX = margin + perLineLength * (i + 1) - targetWH[0] / 2;

            } else if (i + 1 < stepNum) {

                stopX = margin + perLineLength * (i + 1) - passWH[0] / 2;

            } else {

                stopX = margin + perLineLength * (i + 1) - normalWH[0] / 2;

            }

            /*当目标步骤超过 i 时，线条设置为已过颜色，不超过时，设置为普通颜色*/

            if (stepNum > i) {

                linePaint.setColor(passLineColor);

            } else {

                linePaint.setColor(normalLineColor);

            }

            if (isTextBelowLine) {

                /*当文字在线条下方时，设置线条 y 轴的位置并绘制*/

                canvas.drawLine(startX, line2TopMargin, stopX, line2TopMargin, linePaint);

            } else {

                canvas.drawLine(startX, height - line2BottomMargin, stopX, height -
line2BottomMargin, linePaint);

            }

        }

    }
```

3.5.2 画普通步骤点

```
/*绘制一般情况下的步骤点图片*/

private void drawNormalSquar(Canvas canvas, int stepNum) {

    for (int i = 0; i < dotCount; i++) {

        /*在目标点状态时，普通图片不绘制，跳过，继续下一次循环*/

        if (stepNum == i) {

            continue;

        }

        if (stepNum > i) {

            float left = margin + perLineLength * i - passWH[0] / 2;

            float top = 0;

            if (isTextBelowLine) {

                top = line2TopMargin - passWH[1] / 2;

            } else {

                top = height - line2BottomMargin - passWH[1] / 2;

            }

            canvas.drawBitmap(passed_pic, left, top, null);

        } else {

            float left = margin + perLineLength * i - normalWH[0] / 2;

            float top = 0;

            if (isTextBelowLine) {

                top = line2TopMargin - normalWH[1] / 2;

            } else {

                top = height - line2BottomMargin - normalWH[1] / 2;

            }

            canvas.drawBitmap(normal_pic, left, top, null);

        }

    }

}
```

3.5.3 画目标步骤点

```
/*绘制目标步骤图片*/

private void drawTargetSquar(Canvas canvas, int i) {

    float left = margin + perLineLength * i - targetWH[0] / 2;

    float top = 0;

    if (isTextBelowLine) {

        top = line2TopMargin - targetWH[1] / 2;

    } else {

        top = height - line2BottomMargin - targetWH[1] / 2;

    }

    canvas.drawBitmap(target_pic, left, top, null);

}
```

3.5.4 画步骤点说明文字

```
/*绘制各步骤说明文字*/

private void drawDescText(Canvas canvas) {

    for (int i = 0; i < dotCount; i++) {

        String text = texts[i];

        textPaint.getTextBounds(text, 0, text.length(), bounds);

        int textWidth = bounds.width();

        int textHeight = bounds.height();

        float x = margin + perLineLength * i - textWidth / 2;

        float y;

        if (isTextBelowLine) {

            y = height - text2BottomMargin;

        } else {

            y = text2TopMargin + textHeight;

        }

        canvas.drawText(text, x, y, textPaint);
        }

    }
```

通过上面这几个步骤就完成 StepView 的绘制了。

3.6 根据用户设置的是否可点击，给 StepView 添加点击监听

这里先说明一下思路：当用户点击时，View 通过 touch 事件监听用户点击的 x/y 值，然后转换为 point，再通过判断 point 是否在几个步骤点区域范围内，返回对应的步骤点值，然后重新绘制 View。

3.6.1 下面看 onTouchEvent 方法：

```
    @Override

    public boolean onTouchEvent(MotionEvent event) {

        if (clickable) {

            switch (event.getAction()) {

                case MotionEvent.ACTION_DOWN:

                    Point point = new Point();

                    point.x = (int) event.getX();

                    point.y = (int) event.getY();

                    int stepInDots = getStepInDots(point);

                    if (stepInDots != -1) {

                        stepNum = stepInDots;

                        invalidate();

                    }

                    break;

                default:

                    break;

            }

        }

        return super.onTouchEvent(event);

    }
```

3.6.2 获取用户点击点在某个步骤点值：

```
/*获取手指触摸点为第几个步骤点，异常时返回-1*/

private int getStepInDots(Point point) {

    for (int i = 0; i < dotCount; i++) {

        Rect rect = getSetpSquarRects()[i];

        int x = point.x;

        int y = point.y;

        if (rect.contains(x, y)) {

            return i;

        }

    }

    return -1;

}
```

3.6.3 获取各步骤点矩阵所在区域：

```
/*获取所有步骤点的矩阵区域*/

private Rect[] getSetpSquarRects() {

    Rect[] rects = new Rect[dotCount];

    int left, top, right, bottom;

    for (int i = 0; i < dotCount; i++) {

        /*此处默认所有点的区域范围为被选中图片的区域范围*/

        Rect rect = new Rect();

        left = margin + perLineLength * i - targetWH[0] / 2;

        right = margin + perLineLength * i + targetWH[0] / 2;

        if (isTextBelowLine) {

            top = (int) (line2TopMargin - targetWH[1] / 2);

            bottom = (int) (line2TopMargin + targetWH[1] / 2);

        } else {

            top = (int) (height - line2BottomMargin - targetWH[1] / 2);
```

```
                bottom = (int) (height - line2BottomMargin + targetWH[1] / 2);

            }

            rect.set(left, top, right, bottom);

            rects[i] = rect;

        }

        return rects;

    }
```

至此，StepView 的点击事件也完成了。

3.7 设置外部调用接口

```
     /*给外部调用接口，设置步骤总数*/

    public void setDotCount(int count) {

        if (count < 2) {

            throw new IllegalArgumentException("dot count can't be less than 2.");

        }

        dotCount = count;

    }



    /*给外部调用接口，设置说明文字信息*/

    public void setDescription(String[] descs) {

        if (descs == null || descs.length < dotCount) {

            throw new IllegalArgumentException("Descriptions can't be null or its length must 
maore than dot count");

        }

        texts = descs;

    }



    /*给外部调用接口，设置该 view 是否可点击*/

    public void setClickable(boolean clickable) {
```

```
    this.clickable = clickable;

}



/*给外部调用接口，设置步骤*/

public void setStep(Step step) {

    switch (step) {

        case ONE:

            stepNum = 0;

            break;

        case TWO:

            stepNum = 1;

            break;

        case THREE:

            stepNum = 2;

            break;

        case FOUR:

            stepNum = 3;

            break;

        case FIVE:

            stepNum = 4;

            break;

        default:

            break;

    }

    invalidate();

}

/*此处默认最多为5 个步骤*/

public enum Step {
        ONE, TWO, THREE, FOUR, FIVE

    }
```

通过设置这几个接口，可以很方便的动态设置 StepView。

4.部分细节详解

-

详解 1
画线条时，由于目标步骤点比普通的大，因此在计算线条长度时应计算目标步骤点两端线条长度，避免线条长度画错，影响美观。

-

```
            /*设置线条起点 X 轴坐标*/

            if (i == stepNum) {

                startX = margin + perLineLength * i + targetWH[0] / 2;

            } else if (i > stepNum) {

                startX = margin + perLineLength * i + normalWH[0] / 2;

            } else {

                startX = margin + perLineLength * i + passWH[0] / 2;

            }

            /*设置线条终点 X 轴坐标*/

            if (i + 1 == stepNum) {

                stopX = margin + perLineLength * (i + 1) - targetWH[0] / 2;

            } else if (i + 1 < stepNum) {

                stopX = margin + perLineLength * (i + 1) - passWH[0] / 2;

            } else {

                stopX = margin + perLineLength * (i + 1) - normalWH[0] / 2;

            }
```

-

详解 2
线条长度是否可变（见 git view1/view2/view3/view4/view5），当设置线条长度固定时，线条的长度由 view_width/(max_dot-1)决定；当设置线条长度不固定时（view6），由图可知，view6 的长度与 view5 完整的长度一致。

-
-

  详解 3
  文字是否在线条下面，默认是。当文字在线条上面的时候，这里采取数据倒置设置，即把设置给 view 之前的线条距顶部、文字距底部的距离分别设置给了线条距底部、文字距顶部的距离。见如下代码：

-

```
        //当文字在线条上面时，参数倒置

        if (!isTextBelowLine) {

            line2BottomMargin = line2TopMargin;

            text2TopMargin = text2BottomMargin;

        }
```

-

  详解 4
  获取各步骤点的矩形区域，首先是分别对各步骤点区域的左上右下进行计算，然后设置给各步骤点矩形。

-

```
    /*获取所有步骤点的矩阵区域*/

    private Rect[] getSetpSquarRects() {

        Rect[] rects = new Rect[dotCount];

        int left, top, right, bottom;

        for (int i = 0; i < dotCount; i++) {

            /*此处默认所有点的区域范围为被选中图片的区域范围*/

            Rect rect = new Rect();

            left = margin + perLineLength * i - targetWH[0] / 2;

            right = margin + perLineLength * i + targetWH[0] / 2;

            if (isTextBelowLine) {

                top = (int) (line2TopMargin - targetWH[1] / 2);

                bottom = (int) (line2TopMargin + targetWH[1] / 2);

            } else {

                top = (int) (height - line2BottomMargin - targetWH[1] / 2);
```

-

```
                bottom = (int) (height - line2BottomMargin + targetWH[1] / 2);

            }

            rect.set(left, top, right, bottom);

            rects[i] = rect;

        }

        return rects;

    }
```

5.调用

-

  xml 调用

-

```
<com.qb.code.yidian.StepView

      android:id="@+id/step1"

      android:layout_width="match_parent"

      android:layout_height="50dp"

      android:background="#21201D"

      app:count="2"

      app:line_length="fixed_length"

      app:line_stroke_width="1dp"

      app:line_to_top_margin="18dp"

      app:margin="90dp"

      app:max_dot_count="5"

      app:step="one"

      app:text_size="12sp"

      app:text_to_bottom_margin="8dp" />
```

-

  代码调用

-

```
    private StepView step1, step2, step3, step4, step5, step6;

    private CheckBox click1, click2, click3, click4, click5, click6;

    private String[] texts = {"确认身份信息", "确认入住信息", "选择房型", "支付押金", "完成入住
"};



    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);



        step1 = (StepView) findViewById(R.id.step1);

        step2 = (StepView) findViewById(R.id.step2);

        step3 = (StepView) findViewById(R.id.step3);

        step4 = (StepView) findViewById(R.id.step4);

        step5 = (StepView) findViewById(R.id.step5);

        step6 = (StepView) findViewById(R.id.step6);



        click1 = (CheckBox) findViewById(R.id.click1);

        click2 = (CheckBox) findViewById(R.id.click2);

        click3 = (CheckBox) findViewById(R.id.click3);

        click4 = (CheckBox) findViewById(R.id.click4);

        click5 = (CheckBox) findViewById(R.id.click5);

        click6 = (CheckBox) findViewById(R.id.click6);





        step1.setDescription(texts);

        step2.setDescription(texts);

        step3.setDescription(texts);
```

```java
    step4.setDescription(texts);

    step5.setDescription(texts);

    step6.setDescription(texts);



    step1.setStep(StepView.Step.ONE);

    step2.setStep(StepView.Step.TWO);

    step3.setStep(StepView.Step.THREE);

    step4.setStep(StepView.Step.FOUR);

    step5.setStep(StepView.Step.FIVE);

    step6.setStep(StepView.Step.FOUR);



    checkChaged(click1, step1);

    checkChaged(click2, step2);

    checkChaged(click3, step3);

    checkChaged(click4, step4);

    checkChaged(click5, step5);

    checkChaged(click6, step6);

}



private void checkChaged(CheckBox check, final StepView step) {

    check.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

        @Override

        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

            step.setClickable(isChecked);

        }

    });

}
```