# 鸿蒙开源第三方组件——SwipeCaptcha滑动拼图验证组件

## 前言

基于安卓平台的滑动拼图验证组件SwipeCaptcha（  
https://github.com/mcxtzhang/SwipeCaptcha），实现了其核心功能的鸿蒙化迁移和重构，代码已经开源到（https://gitee.com/isrc\_ohos/SwipeCaptcha），欢迎各位下载使用并提出宝贵意见！

## 背景

在页面登录或者注册的时候，为了确保不是机器人操作，会让用户手动验证。验证方式分为滑动拼图验证和滑动验证两种。本文的SwipeCaptcha组件可以实现滑动拼图的验证方式，操作简单，安全性强，被众多APP使用。

## 组件效果展示

鸿蒙系统的SwipeCaptcha组件在使用时，有两个较为重要的图片：滑块和原图。这两张图片被放置于同一水平线上，用户拖动滑块至原图处，误差在一定范围内，即可验证成功。每次调用SwipeCaptcha组件，滑块和原图的位置都会发生随机变化，登录时被暴力破解的难度增加，安全性较高。

在SwipeCaptcha组件的验证界面，还有当前进度值和验证状态的描述。当前进度值表示滑块在水平方向的滑动进度，进度为100时，表示滑块滑至最右端。进度值下方展示的是当前的验证状态，可分为：“开始”、“验证失败，请重新验证三种状态”、“验证成功”。下面依次展示SwipeCaptcha组件拼图验证失败和成功的效果图。

### 验证失败效果

用户未将滑块拖至原图处，导致滑块与原图的位置误差较大，验证失败。

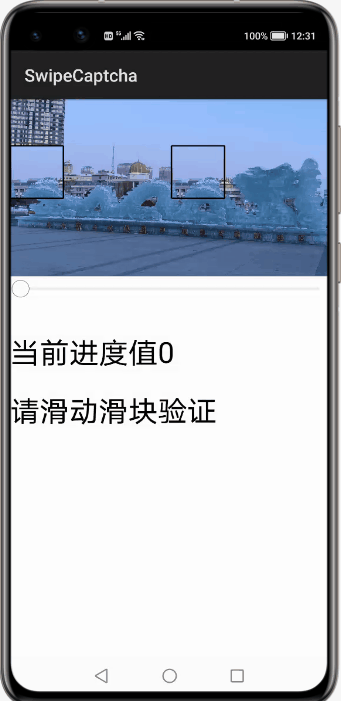


图1 验证失败效果

### 验证成功效果

用户拖动滑块至原图处，误差在一定范围内，验证成功。

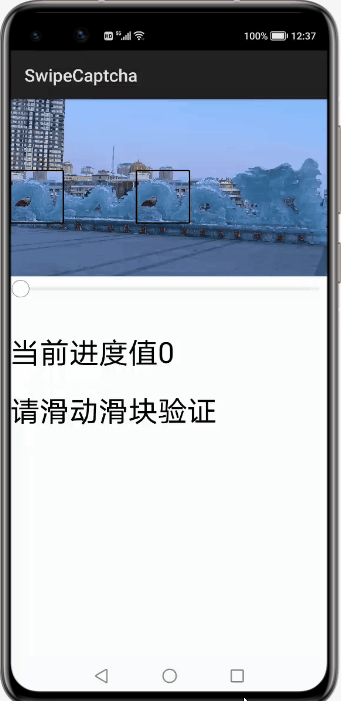


图2 验证成功效果

## Sample解析

Sample主要包含以下四个部分：1）拼图背景导入手机。2）裁剪滑块。3）绘制滑块。4）验证拼图是否成功。下面将通过具体步骤对上述四个部分进行详解。

1. 数据初始化

本步骤包含三个部分的数据设置：（1）获取手机屏幕宽度信息；（2）设置进度值和验证状态的初始提示文字，如“当前进度值”、“请滑动滑块验证”；（3）初始化画笔信息，定义画笔属性；

*//获取手机屏幕宽度displayAttributes.width*DisplayManager displayManager = DisplayManager.getInstance();  
Display display = displayManager.getDefaultDisplay(**this**).get();  
DisplayAttributes displayAttributes = display.getAttributes();  
**windowWidth** = displayAttributes.width;

*// 进度值初始化***text** = **new** Text(**this**);  
**text**.setMarginTop(800);*// 距离顶端边界的距离*  
**text**.setText(**"当前进度值"**+ **progress**);*// 设定文字*  
**text**.setTextSize(100);*// 设定字号*  
**myLayout**.addComponent(**text**);*// 添加进布局中*  
*// 验证状态初始化*

**text2** = **new** Text(**this**);  
**text2**.setMarginTop(1000);  
**text2**.setText(**"请滑动滑块验证"**);  
**text2**.setTextSize(100);  
**myLayout**.addComponent(**text2**);

*//初始化画笔的信息***mPaint** = **new** Paint();  
**mPaint**.setColor(Color.BLACK);*//定义颜色*  
**mPaint**.setAntiAlias(**true**);*//定义虚实线*  
**mPaint**.setStrokeWidth(5f);*//定义宽度*  
**mPaint**.setStyle(Paint.Style.STROKE\_STYLE);*//定义绘图方式*

1. 背景图片绘制

用手机屏幕的宽度除以背景图片的宽度，得到背景图片的缩放比例，当该图片显示在手机中，按照此比例缩放可与屏幕同宽。该比例用于背景图片适配不同型号的手机屏幕。

*//背景图片的缩放比例***float** ratio = (**float**) **windowWidth**/(**float**) img.getImageInfo().size.width;

*//背景图片绘制*Component image = **new** Component(**this**);  
Component.DrawTask drawTask = **new** Component.DrawTask() {  
 @Override  
 **public void** onDraw(Component component, Canvas canvas) {  
  *//按照比例进行缩放* canvas.scale(ratio , ratio);  
  *//绘图*

canvas.drawPixelMapHolder(pixelMapHolder, 0, 0, **new** Paint());  
 }  
};  
image.addDrawTask(drawTask);  
**myLayout**.addComponent(image);

3. 确定滑块和原图的位置

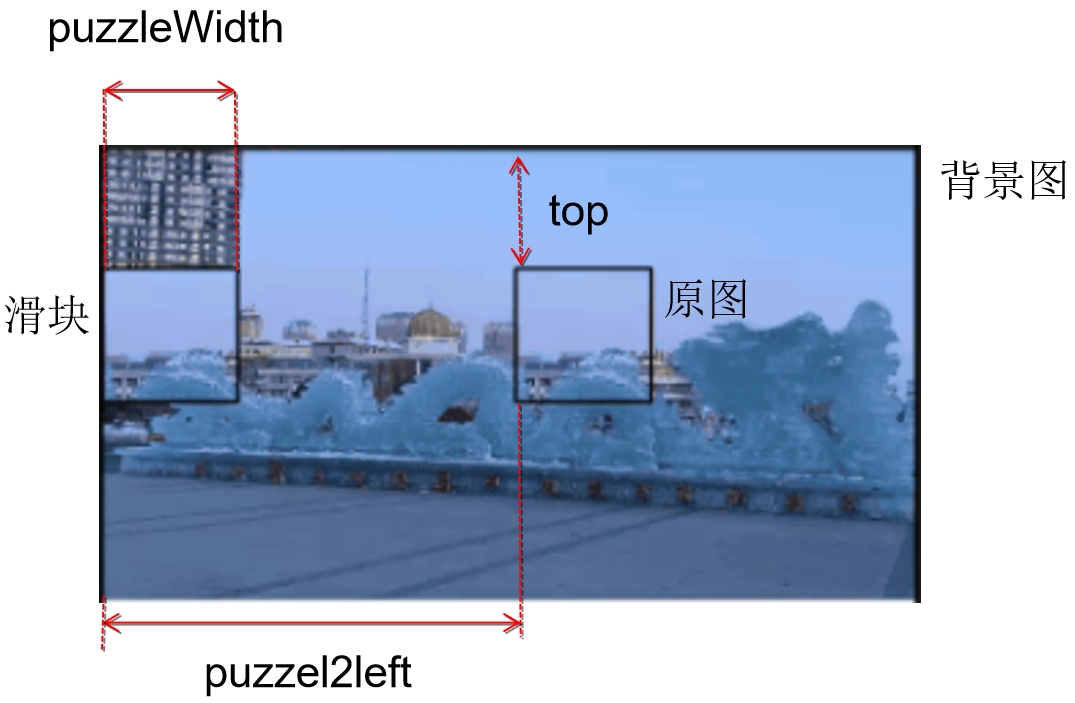


图3 滑块和原图的位置示意

puzzleWidth为滑块或者原图的宽度；top为随机数值，表示滑块或者原图的上边距离背景图片上边的距离；puzzel2left也为随机数值，表示原图左边距离背景图片左边的距离。有了以上三个变量可以确定组件中滑块和原图的初始位置和大小（滑块初始时位于屏幕的最左侧）。下面介绍上述属性是如何计算出来的。

*//puzzleWidth为屏幕宽度的1/6*

**puzzleWidth** = **windowWidth**/6;

*//top为图片缩放后高度与抠图高度之差再乘以随机数***top** = (**float**) Math.random()\*(img.getImageInfo().size.height\*ratio - **puzzleWidth**);

*//原图位置一定在滑块位置右面*

*//屏幕宽度减去两个拼图宽度 \*随机数，后向右平移一个滑块的长度*

**puzzel2left** = ((**windowWidth** -**puzzleWidth**\*2) \* (**float**)Math.random()) + **puzzleWidth**;

4. 获取滑块  
 本步骤需要根据原图的位置，解码出一个图片作为滑块。首先设置滑块的形状为矩形，依据上述的puzzel2left、puzzleWidth属性，确定矩形所在区域，依据缩放比例，将矩形区域映射为原比例图像，并对此图像进行解码，得到滑块图像数据。

PixelMap puzzlePixelMap = *getPuzzlePixelMap*(**this** , ResourceTable.***Media\_longa*** , **new** Rect((**int**)(**puzzel2left**/ratio), (**int**) (**top**/ratio), (**int**) (**puzzleWidth**/ratio) , (**int**) (**puzzleWidth**/ratio)));  
PixelMapHolder pixelMapHolder1 = **new** PixelMapHolder(puzzlePixelMap);

5.绘制滑块

滑块通过画笔来绘制，其位置应该根据滑动进度条的进度来移动，并且要对不同手机屏幕的大小进行适配。同时，为了和用户友好的交互，我们还需要为滑块绘制一个边框，告知用户这个边框所在就是滑块（原图也需要绘制边框，原理相同）。绘制滑块和边框的代码如下：

*//绘制滑块*Component.DrawTask puzzelDrawTask = **new** Component.DrawTask() {  
 @Override  
 **public void** onDraw(Component component, Canvas canvas) {  
 Paint paint = **new** Paint();

*//移动小滑块拼图*  
 canvas.translate(**slider**.getProgress()\*displayAttributes.width /100 , **top**);

*//进行适当比例缩放* canvas.scale(ratio , ratio);  
 canvas.drawPixelMapHolder(pixelMapHolder1 , 0 , 0 , paint);  
  
 }  
};

*//绘制滑块边框*

Component puzzleFrame = **new** Component(**this**);  
Component.DrawTask drawTask2 = **new** Component.DrawTask() {  
 @Override  
 **public void** onDraw(Component component, Canvas canvas) {  
  *//方框左侧位置*

**float** left = **slider**.getProgress()\***windowWidth** /100;  
 *//绘制边框的左边*

canvas.drawLine(**new** Point(left , **top**),  
 **new** Point(left, **top** + **puzzleWidth**), **mPaint**);  
  *//绘制边框的上边*

canvas.drawLine(**new** Point(left, **top**),  
 **new** Point(left + **puzzleWidth**, **top**), **mPaint**);  
  *//绘制边框的右边* canvas.drawLine(**new** Point(left + **puzzleWidth**, **top**),  
 **new** Point(left + **puzzleWidth**, **top** + **puzzleWidth**), **mPaint**);  
  *//绘制边框的下边*

canvas.drawLine(**new** Point(left, **top** + **puzzleWidth**),  
 **new** Point(left + **puzzleWidth**, **top** + **puzzleWidth**), **mPaint**);  
 }  
};

6. 进度条滑动更新

为进度条设置监听，拖动进度条会引起三处更新：（1）滑块位置和滑块边框位置的更新；（2）进度值的更新；（3）验证状态的更新。在验证状态的更新中，需要对用户拖动进度条结束时的验证状态进行判断，滑块和原图的位置差距是否在误差范围内，如果在范围内，则显示验证成功，如果不在误差范围内，则显示验证失败，提示需要重新验证。

*//设置进度条监听*

**slider**.setValueChangedListener(**new** Slider.ValueChangedListener() {  
 @Override

*//拖动进度条引起的更新*

**public void** onProgressUpdated(Slider slider, **int** i, **boolean** b) { *//滑块的位置更新* **puzzle**.invalidate();  
  *//滑块边框位置的更新* puzzleFrame.invalidate();  
  *//进度值更新*

**text**.setText(**"当前进度值 : "** + slider.getProgress());  
 }

}

*//当用户开始滑动进度条时，验证状态变为“开始”字样。*

**public void** onTouchStart(Slider slider) {  
  *//开始拖动的方法* **text2**.setText(**"开始"**);  
}

*//判断滑块左侧边的位置和原图的左侧边的位置是否在误差内*

**public void** onTouchEnd(Slider slider) {  
 **if**(((slider.getProgress()\***windowWidth** /100)<(**puzzel2left** + **puzzleWidth**/10))&&((slider.getProgress()\***windowWidth** /100)>(**puzzel2left** - **puzzleWidth**/10)))  
 {  
 **text2**.setText(**"验证成功"**);  
 }**else** {  
 **text2**.setText(**"验证失败，请重新验证"**);  
 slider.setProgressValue(10);  
 }  
}

## 项目贡献人

赵柏屹 郑森文 朱伟 陈美汝 张馨心