滴滴的 “哆啦A梦” DoKit 背后的技术实现 1

嘟嚕 鸿洋 11月15日

本文作者

作者：**嘟嚕**

链接：

https://juejin.im/post/5c4dcfe8518825261e1f2978

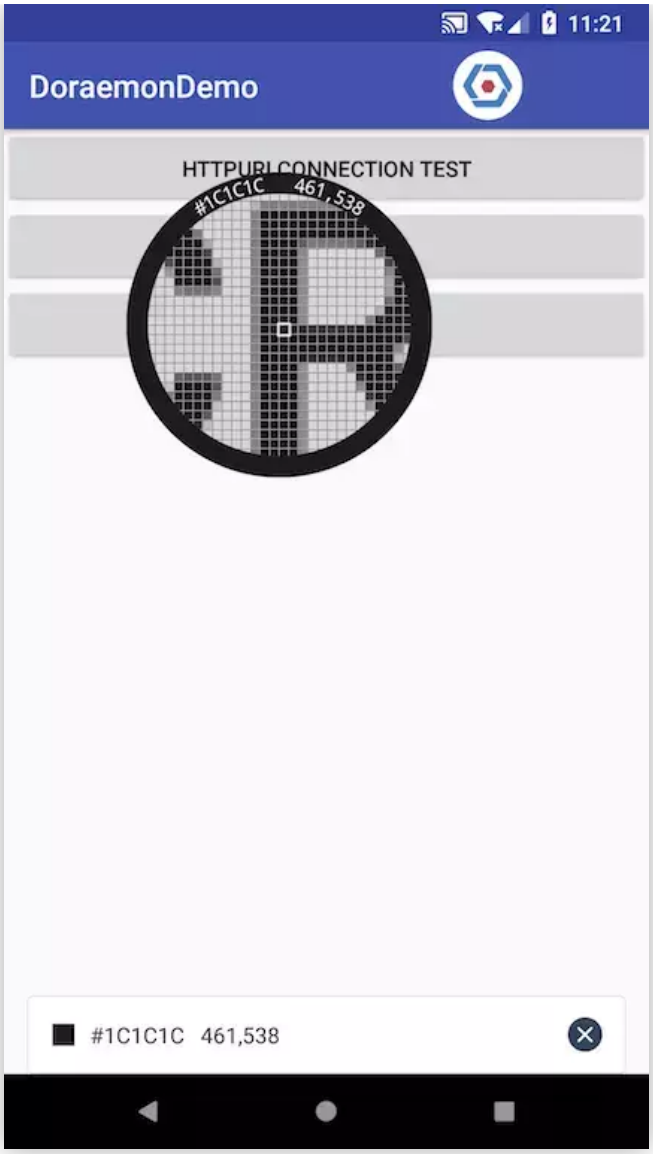
> DoraemonKit /'dɔ:ra:'emɔn/，简称DoKit，中文名 哆啦A梦，意味着能够像哆啦A梦一样提供给他的主人各种各样的工具。Just Do Kit
>
> DoraemonKit 是一个功能集合面板，能够让每一个 App 快速接入一些常用的或者你没有实现的一些辅助开发工具、测试效率工具、视觉辅助工具，而且能够完美在 Doraemon 面板中接入你

1 引言

DoraemonKit是滴滴开源的研发助手组件，目前支持iOS和Android两个平台。通过接入DoraemonKit组件，可以方便支持如下所示的多种调试工具：

本文是DoraemonKit之Android版本技术实现系列文章的第一篇，主要介绍各个视觉工具的技术实现细节。

2 技术实现：取色器



方案对比

取色器工具可以通过颜色吸管获取屏幕任意位置的像素值，所以实现的关键就是如何获取像素点。获取像素点的第一步是获取屏幕截图，获取屏幕截图在Android平台主要有以下几种方式：

1. 通过View的getDrawingCache方法
2. 通过读取系统FrameBuffer
3. 通过MediaProjectionManager类

对比三种实现方式：

方式一只能获取当前Window内DocorView的内容，不能获取状态栏或者脱离应用本身，且开启DrawingCache会增加应用内存占用；

方式二中FrameBuffer不能直接读取，需要获得系统Root权限，且兼容性差；

方式三可脱离应用本身获取应用外截屏，截图取自系统Binder不占用应用内存，只需请求录屏权限。

	getDrawingCache函数	读取系统FrameBuffer	MediaProjectionManager类
实现复杂度	简单	复杂	较简单
需要权限	无	Root权限	录屏权限
适用性	只能截取应用内	应用内外都支持	应用内外都支持
性能影响	大	小	小

通过对比，DoraemonKit选择方式三作为取色器的实现方案。

请求录屏权限

```
private boolean requestCaptureScreen() {
  if (android.os.Build.VERSION.SDK_INT < android.os.Build.VERSION_CODES.LOLLIPOP) {
    return false;
  }
  MediaProjectionManager mediaProjectionManager = (MediaProjectionManager) getContext().getSystemService(Context.MEDIA
  if (mediaProjectionManager == null) {
    return false;
  }
  startActivityForResult(mediaProjectionManager.createScreenCaptureIntent(), RequestCode.CAPTURE_SCREEN);
  return true;
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  if (requestCode == RequestCode.CAPTURE_SCREEN && resultCode == Activity.RESULT_OK) {
    showColorPicker(data);
    ...
  } else {
    ...
  }
}
```

通过createScreenCaptureIntent()方法可以获取请求系统录屏权限的Intent，然后调用startActivityForResult，系统会自动弹出权限授予弹窗。如果授予权限则在onActivityResult中得到系统回调

成功，且返回录屏的resultData。

创建ImageReader

```
public void init(Context context, Bundle bundle) {
  mMediaProjectionManager = (MediaProjectionManager) context.getSystemService(Context.MEDIA_PROJECTION_SERVICE);
  if (mMediaProjectionManager != null) {
    Intent intent = new Intent();
    intent.putExtras(bundle);
    mMediaProjection = mMediaProjectionManager.getMediaProjection(Activity.RESULT_OK, intent);
  }
  int width = UIUtils.getWidthPixels(context);
  int height = UIUtils.getRealHeightPixels(context);
  int dpi = UIUtils.getDensityDpi(context);
  mImageReader = ImageReader.newInstance(width, height, PixelFormat.RGBA_8888, 2);
  mMediaProjection.createVirtualDisplay("ScreenCapture",
      width, height, dpi,
      DisplayManager.VIRTUAL_DISPLAY_FLAG_AUTO_MIRROR,
      mImageReader.getSurface(), null, null);
}
```

通过onActivityResult中返回的resultData就可以创建系统录屏服务MediaProjection，然后创建ImageReader并与MediaProjection的Surface进行绑定，之后就可以通过ImageReader获取屏幕截图了。

获取屏幕截图和像素点

```
public void capture() {
  if (isCapturing) {
    return;
  }
  isCapturing = true;
  Image image = mImageReader.acquireLatestImage();
  if (image == null) {
    return;
  }
  int width = image.getWidth();
  int height = image.getHeight();
  Image.Plane[] planes = image.getPlanes();
  ByteBuffer buffer = planes[0].getBuffer();
  int pixelStride = planes[0].getPixelStride();
  int rowStride = planes[0].getRowStride();
  int rowPaddingStride = rowStride - pixelStride * width;
  int rowPadding = rowPaddingStride / pixelStride;
  Bitmap recordBitmap = Bitmap.createBitmap(width + rowPadding , height, Bitmap.Config.ARGB_8888);
  recordBitmap.copyPixelsFromBuffer(buffer);
  mBitmap = Bitmap.createBitmap(recordBitmap, 0, 0, width, height);
  image.close();
  isCapturing = false;
}
```

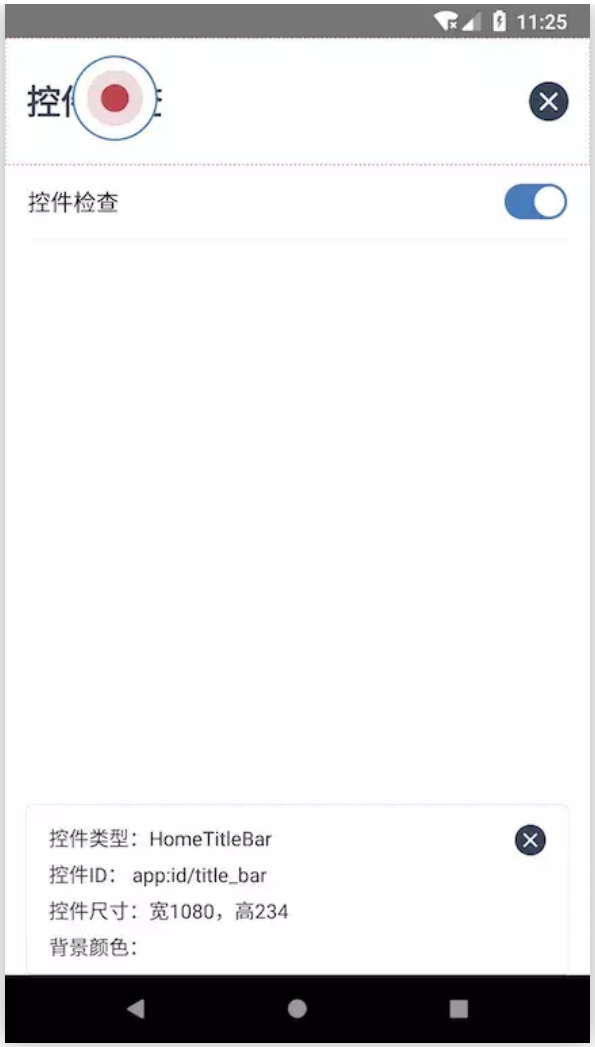
调用ImageReader的acquireLatestImage可以获取当前屏幕的截图，然后将Image对象转为Bitmap对象就可以方便地进行显示和获取像素点了。

```
public static int getPixel(Bitmap bitmap, int x, int y) {
  if (bitmap == null) {
    return -1;
  }
  if (x < 0 || x > bitmap.getWidth()) {
    return -1;
  }
  if (y < 0 || y > bitmap.getHeight()) {
    return -1;
  }
  return bitmap.getPixel(x, y);
}
```

根据浮标的坐标在Bitmap上就可以很方便地获取像素点的值了。

3 技术实现：控件检查

控件检查的功能是通过浮标选取目标View，然后获取目标View的相关信息，所以如何获取这个View的引用就是实现这一功能的关键。



监听前台Activity的进入

```
app.registerActivityLifecycleCallbacks(new Application.ActivityLifecycleCallbacks() {
        @Override
      public void onActivityResumed(Activity activity) {
        ...
        for (ActivityLifecycleListener listener : sListeners) {
          listener.onActivityResumed(activity);
        }
      }
}
```

通过注册监听可以在Activity进入Resumed状态时获得通知，这样在浮标移动时DoraemonKit就可以持有最前台的Activity。

遍历ViewTree获取目标View

```
private View traverseViews(View view, int x, int y) {
  int[] location = new int[2];
```

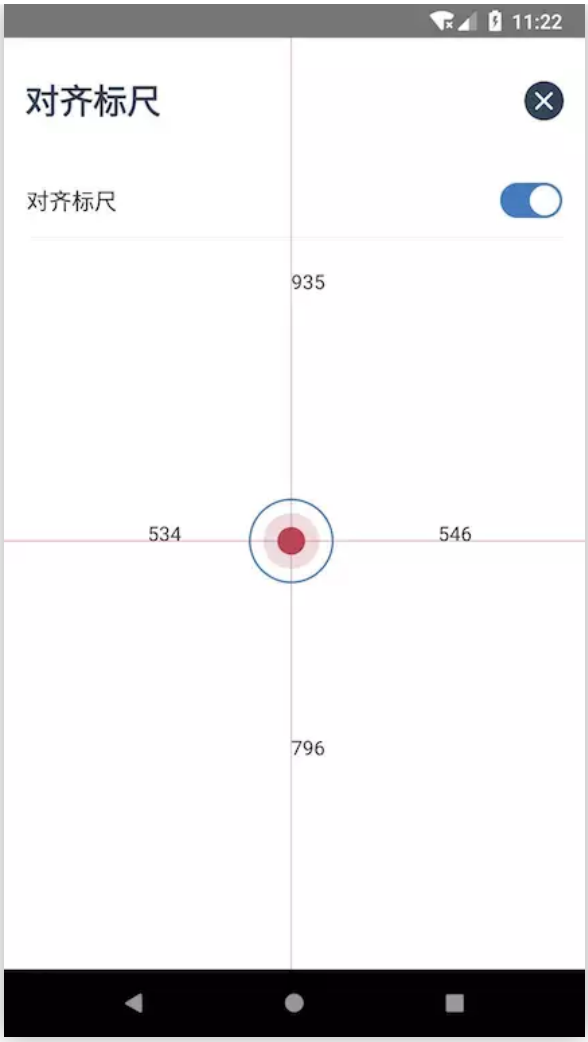
```
    view.getLocationInWindow(location);
    int left = location[0];
    int top = location[1];
    int right = left + view.getWidth();
    int bottom = top + view.getHeight();
    if (view instanceof ViewGroup) {
      int childCount = ((ViewGroup) view).getChildCount();
      if (childCount != 0) {
        for (int index = childCount - 1; index >= 0; index--) {
          View v = traverseViews(((ViewGroup) view).getChildAt(index), x, y);
          if (v != null) {
            return v;
          }
        }
      }
      if (left < x && x < right && top < y && y < bottom) {
        return view;
      } else {
        return null;
      }
    } else {
      LogHelper.d(TAG, "class: " + view.getClass() + ", left: " + left
          + ", right: " + right + ", top: " + top + ", bottom: " + bottom);
      if (left < x && x < right && top < y && y < bottom) {
        return view;
      } else {
        return null;
      }
    }
  }
```

因为View是以Tree的结构组织的，所以通过遍历当前Activity的ViewTree就可以获取到目标View。以DocorView作为根，递归调用同时判断浮标坐标是否在View的范围即可以得到目标View。

因为View可能存在覆盖关系，所以需要使用深度优先遍历才能获得最顶端的View。

4 技术实现：对齐标尺

对齐标尺的实现比较简单，只需要根据浮标坐标绘制水平标尺线和竖直标尺线即可，实现效果如下图。



5 技术实现：布局边界

布局边界

DoraemonKit最开始实现布局边界，是通过遍历ViewTree在悬浮Window上绘制边框线的方式，但这种方式有一个问题就是如果Activity包含多个层级的时候，所有层级View的边框都会被绘制在最顶层，导致显示十分混乱，在复杂界面上基本是不可用的。

在调研了几种方式之后，DoraemonKit使用了替换View的Background的方式。View的Background是Drawable类型的，而LayerDrawable这种Drawable是可以包含一组Drawable的，所以取出View的原始Background后与绘制边框的Drawable放进同一个LayerDrawable中，就可以实现带边框的背景。

```
private void traverseChild(View view) {
  if (view instanceof ViewGroup) {
```

```
      replaceDrawable(view);
      int childCount = ((ViewGroup) view).getChildCount();
      if (childCount != 0) {
        for (int index = 0; index < childCount; index++) {
          traverseChild(((ViewGroup) view).getChildAt(index));
        }
      }
    } else {
      replaceDrawable(view);
    }
  }

  private void replaceDrawable(View view) {
    LayerDrawable newDrawable;
    if (view.getBackground() != null) {
      Drawable oldDrawable = view.getBackground();
      if (oldDrawable instanceof LayerDrawable) {
        for (int i = 0; i < ((LayerDrawable) oldDrawable).getNumberOfLayers(); i++) {
          if (((LayerDrawable) oldDrawable).getDrawable(i) instanceof ViewBorderDrawable) {
            // already replace
            return;
          }
        }
        newDrawable = new LayerDrawable(new Drawable[] {
            oldDrawable,
            new ViewBorderDrawable(view)
        });
      } else {
        newDrawable = new LayerDrawable(new Drawable[] {
            oldDrawable,
            new ViewBorderDrawable(view)
        });
      }
    } else {
      newDrawable = new LayerDrawable(new Drawable[] {
          new ViewBorderDrawable(view)
      });
    }
    view.setBackground(newDrawable);
  }
```

这种方式的好处就是实现简单，且兼容性很好，不会出现显示异常，包括多个层级的覆盖也能正常显示，能实时地伴随组件隐藏和显示，但也存在一定的侵入性，会对View的绘制造成一定的开销。

布局层级

布局层级功能可以很方便地查看当前页面的Layout层级，是一个3D可视化的效果，可以多个角度旋转查看，这个功能是依赖jakewharton的scalpel项目实现的，这个项目的核心只有一个源码文件ScalpelFrameLayout，通过把希望查看层级的页面根View加到这个Layout中就可以实现布局层级功能。

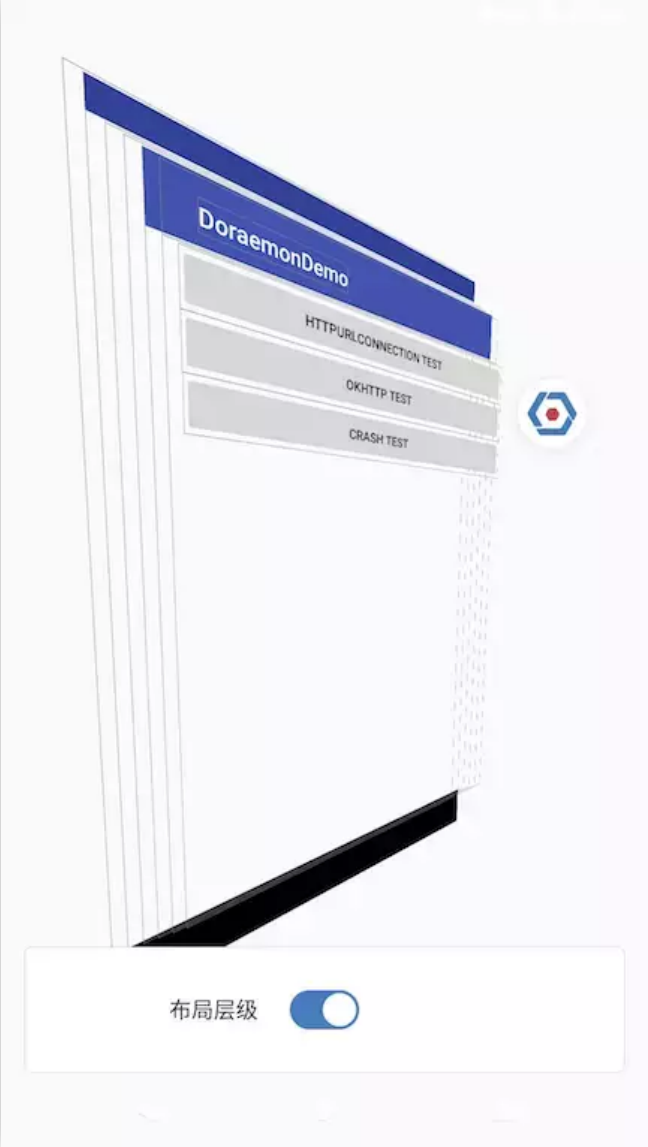
这个Layout的实现原理是重写了Layout的draw方法，通过Camera进行了3D变换，重新进行了排布绘制。

```
@Override public void draw(@SuppressWarnings("NullableProblems") Canvas canvas) {
  if (!enabled) {
    super.draw(canvas);
    return;
  }

  getLocationInWindow(location);
  float x = location[0];
  float y = location[1];

  int saveCount = canvas.save();
```

```
float cx = getWidth() / 2f;
float cy = getHeight() / 2f;

camera.save();
camera.rotate(rotationX, rotationY, 0);
camera.getMatrix(matrix);
camera.restore();

matrix.preTranslate(-cx, -cy);
matrix.postTranslate(cx, cy);
canvas.concat(matrix);
canvas.scale(zoom, zoom, cx, cy);

if (!layeredViewQueue.isEmpty()) {
  throw new AssertionError("View queue is not empty.");
}

// We don't want to be rendered so seed the queue with our children.
for (int i = 0, count = getChildCount(); i < count; i++) {
  LayeredView layeredView = layeredViewPool.obtain();
  layeredView.set(getChildAt(i), 0);
  layeredViewQueue.add(layeredView);
}

while (!layeredViewQueue.isEmpty()) {
  LayeredView layeredView = layeredViewQueue.removeFirst();
  View view = layeredView.view;
  int layer = layeredView.layer;

  // Restore the object to the pool for use later.
  layeredView.clear();
  layeredViewPool.restore(layeredView);

  // Hide any visible children.
  if (view instanceof ViewGroup) {
    ViewGroup viewGroup = (ViewGroup) view;
    visibilities.clear();
    for (int i = 0, count = viewGroup.getChildCount(); i < count; i++) {
      View child = viewGroup.getChildAt(i);
      //noinspection ConstantConditions
      if (child.getVisibility() == VISIBLE) {
        visibilities.set(i);
        child.setVisibility(INVISIBLE);
      }
    }
  }

  int viewSaveCount = canvas.save();

  // Scale the layer index translation by the rotation amount.
  float translateShowX = rotationY / ROTATION_MAX;
  float translateShowY = rotationX / ROTATION_MAX;
  float tx = layer * spacing * density * translateShowX;
  float ty = layer * spacing * density * translateShowY;
  canvas.translate(tx, -ty);

  view.getLocationInWindow(location);
  canvas.translate(location[0] - x, location[1] - y);

  viewBoundsRect.set(0, 0, view.getWidth(), view.getHeight());
  canvas.drawRect(viewBoundsRect, viewBorderPaint);

  if (drawViews) {
    if (!(view instanceof SurfaceView)) {
      view.draw(canvas);
```

```
        }
      }

      if (drawIds) {
        int id = view.getId();
        if (id != NO_ID) {
          canvas.drawText(nameForId(id), textOffset, textSize, viewBorderPaint);
        }
      }

      canvas.restoreToCount(viewSaveCount);

      // Restore any hidden children and queue them for later drawing.
      if (view instanceof ViewGroup) {
        ViewGroup viewGroup = (ViewGroup) view;
        for (int i = 0, count = viewGroup.getChildCount(); i < count; i++) {
          if (visibilities.get(i)) {
            View child = viewGroup.getChildAt(i);
            //noinspection ConstantConditions
            child.setVisibility(VISIBLE);
            LayeredView childLayeredView = layeredViewPool.obtain();
            childLayeredView.set(child, layer + 1);
            layeredViewQueue.add(childLayeredView);
          }
        }
      }
    }

    canvas.restoreToCount(saveCount);
  }
```

布局层级在添加SurfaceView等特殊绘制的View时可能出现绘制问题，出现黑白屏闪烁问题，需要屏蔽这些特殊View的绘制。



6 总结

取色器组件的实现主要通过系统录屏api，从截图Bitmap中取得像素点。

控件检查功能通过遍历ViewTree实现，需要注册全局Activity的生命周期监听。

对齐标尺功能直接通过浮标的屏幕坐标绘制水平和垂直标尺。

布局边界功能通过替换View的Background实现，由包含原始Background的LayerDrawable替换原有Background。

布局层级主要是使用开源项目scalpel实现，对原有ViewTree进行3D变换，重新进行绘制。

通过这篇文章主要是希望大家能够对DoraemonKit视觉工具的技术实现有一个了解，如果有好的想法也可以参与到DoraemonKit开源项目的建设中来，在项目页面提交Issues或者提交Pull Requests，相信DoraemonKit项目在大家的努力下会越来越完善。

DoraemonKit项目地址：
https://github.com/didi/DoraemonKit

觉得不错的话就给项目点个star吧。

推荐阅读：

怎么阻止同事在代码里面“下毒”？ | 流畅度篇

小缘你咋学的Android？为啥我们不一样~

滴滴的开源的Booster分析 是如何修复系统bug的？

扫一扫 关注我的公众号

如果你想要跟大家分享你的文章，欢迎投稿~

┏(^ 0 ^)┛明天见！

阅读原文