深入解析 Android 中的自定义绘图

本博客文章在 *ChatGPT-4o* 的帮助下编写。

介绍

在这篇博客中，我们将探讨 `DrawActivity` 类，这是一个在 Android 应用中实现自定义绘图视图的全面示例。我们将分解每个组件和使用的算法，详细解释它们如何协同工作以实现所需的功能。

目录

DrawActivity 概述

`DrawActivity` 是处理绘图操作、图像裁剪以及与其他组件（如 fragment 和图像上传）交互的主要活动。它提供了一个用户界面，用户可以在其中绘图、撤销、重做和操作图像。

```
public class DrawActivity extends Activity implements View.OnClickListener {
  // 请求代码和 fragment ID 的常量
  public static final int CAMERA_RESULT = 1;
  public static final int CROP_RESULT = 2;
  public static final int DRAW_FRAGMENT = 0;
  public static final int RECOG_FRAGMENT = 1;
  public static final int RESULT_FRAGMENT = 2;
  public static final int WAIT_FRAGMENT = 3;
  public static final int MATERIAL_RESULT = 4;
  public static final String RESULT_JSON = "resultJson";
```

```
  public static final int INIT_FLOWER_ID = R.drawable.flower_b;
  public static final int LOGOUT = 0;
  public static final int IMAGE_RESULT = 0;

  // 处理图像和绘图操作的变量
  String baseUrl;
  DrawView drawView;
  Bitmap originImg;
  public static DrawActivity instance;
  View dir, clear, cameraView, materialView, scale;
  ImageView undoView, redoView;
  View upload;
  String cropPath;
  Tooltip toolTip;
  int curFragmentId = -1;
  int serverId = -1;
  private Bitmap resultBitmap;
  private RadioGroup radioGroup;
  Fragment curFragment;
  int curDrawMode;
  RadioButton drawBackBtn;
  private Activity cxt;
  Uri curPicUri;
}
```

初始化 Activity

在 Activity 创建时，执行各种初始化操作，如设置视图、加载初始图像和配置事件监听器。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  instance = this;
  cxt = this;
  cropPath = PathUtils.getCropPath();
  setContentView(R.layout.draw_layout);
```

```
  findView();
  setSize();
  initOriginImage();
  toolTip = new Tooltip(this);
  initUndoRedoEnable();
  setIp();
  initDrawmode();
}
```

findView()

此方法初始化 Activity 中使用的视图。

```
private void findView() {
  drawView = findViewById(R.id.drawView);
  undoView = findViewById(R.id.undo);
  redoView = findViewById(R.id.redo);
  scale = findViewById(R.id.scale);
  upload = findViewById(R.id.upload);
  clear = findViewById(R.id.clear);
  dir = findViewById(R.id.dir);
  materialView = findViewById(R.id.material);
  cameraView = findViewById(R.id.camera);

  dir.setOnClickListener(this);
  materialView.setOnClickListener(this);
  undoView.setOnClickListener(this);
  scale.setOnClickListener(this);
  redoView.setOnClickListener(this);
  clear.setOnClickListener(this);
  cameraView.setOnClickListener(this);
  upload.setOnClickListener(this);
  initRadio();
}
```

setSize()

设置绘图视图的大小。

```
private void setSize() {
  setSizeByResourceSize();
```

```
    setViewSize(drawView);
}

private void setSizeByResourceSize() {
    int width = getResources().getDimensionPixelSize(R.dimen.draw_width);
    int height = getResources().getDimensionPixelSize(R.dimen.draw_height);
    App.drawWidth = width;
    App.drawHeight = height;
}

private void setViewSize(View v) {
    ViewGroup.LayoutParams lp = v.getLayoutParams();
    lp.width = App.drawWidth;
    lp.height = App.drawHeight;
    v.setLayoutParams(lp);
}
```

initOriginImage()
加载将用于绘图的初始图像。

```
private void initOriginImage() {
    Bitmap bitmap = BitmapFactory.decodeResource(getResources(), INIT_FLOWER_ID);
    String imgPath = PathUtils.getCameraPath();
    BitmapUtils.saveBitmapToPath(bitmap, imgPath);
    Uri uri1 = Uri.fromFile(new File(imgPath));
    setImageByUri(uri1);
}
```

处理图像操作

Activity 处理各种图像操作，如通过 URI 设置图像、裁剪和保存绘制的位图。

setImageByUri(Uri uri)
从给定的 URI 加载图像并准备绘图。

```
private void setImageByUri(final Uri uri) {
    new Handler().postDelayed(new Runnable() {
        @Override
```

```
    public void run() {
      curPicUri = uri;
      Bitmap bitmap = null;
      try {
        if (uri != null) {
          bitmap = BitmapUtils.getBitmapByUri(DrawActivity.this, uri);
        }
      } catch (Exception e) {
        e.printStackTrace();
      }

      int originW = bitmap.getWidth();
      int originH = bitmap.getHeight();
      if (originW != App.drawWidth || originH != App.drawHeight) {
        float originRadio = originW * 1.0f / originH;
        float radio = App.drawWidth * 1.0f / App.drawHeight;
        if (Math.abs(originRadio - radio) < 0.01) {
          Bitmap originBm = bitmap;
          bitmap = Bitmap.createScaledBitmap(originBm, App.drawWidth, App.drawHeight, false);
          originBm.recycle();
        } else {
          cropIt(uri);
          return;
        }
      }
      ImageLoader imageLoader = ImageLoader.getInstance();
      imageLoader.addOrReplaceToMemoryCache("origin", bitmap);
      originImg = bitmap;
      serverId = -1;

      drawView.setSrcBitmap(originImg);
      showDrawFragment(App.ALL_INFO);
      curDrawMode = App.DRAW_FORE;
    }
  }, 500);
}
```

cropIt(Uri uri)

启动图像裁剪活动。

```
public void cropIt(Uri uri) {
  Crop.startPhotoCrop(this, uri, cropPath, CROP_RESULT);
}
```

saveBitmap()

将绘制的位图保存到文件并上传到服务器。

```
public void saveBitmap() {
  Bitmap handBitmap = drawView.getHandBitmap();
  Bitmap originBitmap = drawView.getSrcBitmap();
  saveBitmapToFileAndUpload(handBitmap, originBitmap);
}
```

saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap)

将位图保存到文件并异步上传。

```
private void saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap) {
  final String originPath = PathUtils.getOriginPath();
  BitmapUtils.saveBitmapToPath(originBitmap, originPath);
  final String handPath = PathUtils.getHandPath();
  BitmapUtils.saveBitmapToPath(handBitmap, handPath);
  new AsyncTask<Void, Void, Void>() {
    boolean res;
    Bitmap foreBitmap;
    Bitmap backBitmap;

    @Override
    protected void onPreExecute() {
      super.onPreExecute();
      showWaitFragment();
    }

    @Override
    protected Void doInBackground(Void... params) {
      try {
        if (baseUrl == null) {
```

```
        throw new Exception("baseUrl is null");
      }
      String jsonRes = UploadImage.upload(baseUrl, serverId, Web.STATUS_CONTINUE, originPath, handPat
      getJsonData(jsonRes);
      res = true;
    } catch (Exception e) {
      res = false;
      e.printStackTrace();
    }
    return null;
  }

  private void getJsonData(String jsonRes) throws Exception {
    JSONObject json = new JSONObject(jsonRes);
    if (serverId == -1) {
      serverId = json.getInt(Web.ID);
    }
    String foreUrl = json.getString(Web.FORE);
    String backUrl = json.getString(Web.BACK);
    String resultUrl = json.getString(Web.RESULT);
    foreBitmap = Web.getBitmapFromUrlByStream1(foreUrl, 0);
    backBitmap = Web.getBitmapFromUrlByStream1(backUrl, 0);
    resultBitmap = Web.getBitmapFromUrlByStream1(resultUrl, 0);
  }

  @Override
  protected void onPostExecute(Void aVoid) {
    super.onPostExecute(aVoid);
    if (res) {
      showRecogFragment(foreBitmap, backBitmap);
    } else {
      Utils.toast(DrawActivity.this, R.string.server_error);
      recogNo();
    }
  }
```

```
  }.execute();
}
```

Fragment 管理

Activity 管理不同的 fragment 以处理应用的各种状态，如绘图、识别和等待。

showDrawFragment(int infoId)
显示绘图 fragment。

```
private void showDrawFragment(int infoId) {
  curFragmentId = DRAW_FRAGMENT;
  curFragment = new DrawFragment(infoId);
  showFragment(curFragment);
}
```

showWaitFragment()
显示等待 fragment。

```
private void showWaitFragment() {
  curFragmentId = WAIT_FRAGMENT;
  showFragment(new WaitFragment());
}
```

showFragment(Fragment fragment)
用指定的 fragment 替换当前 fragment。

```
private void showFragment(Fragment fragment) {
  FragmentTransaction trans = getFragmentManager().beginTransaction();
  trans.replace(R.id.rightLayout, fragment);
  trans.commit();
}
```

事件处理

Activity 处理各种用户交互，如按钮点击和菜单选择。

onClick(View v)
处理不同视图的点击事件。

```
@Override
public void onClick(View v) {
  int id = v.getId();
  if (id == R.id.drawOk) {
    if (drawView.isDrawFinish()) {
      saveBitmap();
    } else {
      Utils.alertDialog(this, R.string.please_draw_finish);
    }
  } else if (id == R.id.recogOk) {
    recogOk();
  } else if (id == R.id.recogNo) {
    recogNo();
  } else if (id == R.id.dir) {
    Utils.getGalleryPhoto(this, IMAGE_RESULT);
  } else if (id == R.id.clear) {
    clearEverything();
  } else if (id == R.id.undo) {
    drawView.undo();
  } else if (id == R.id.redo) {
    drawView.redo();
  } else if (id == R.id.camera) {
    Utils.takePhoto(cxt, CAMERA_RESULT);
  } else if (id == R.id.material) {
    goMaterial();
  } else if (id == R.id.upload) {
    com.lzw.commons.Utils.goActivity(cxt, PhotoActivity.class);
  } else if (id == R.id.scale) {
    cropIt(curPicUri);
  }
}
```

onActivityResult(int requestCode, int resultCode, Intent data)
处理其他活动的结果，如图像选择或裁剪。

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
  if (resultCode != RESULT_CANCELED) {
    Uri uri;
    switch (requestCode) {
      case IMAGE_RESULT:
        if (data != null) {
          setImageByUri(data.getData());
        }
        break;
      case CAMERA_RESULT:
        setImageByUri(Utils.getCameraUri());
        break;
      case CROP_RESULT:
        uri = Uri.fromFile(new File(cropPath));
        setImageByUri(uri);
        break;
      case MATERIAL_RESULT:
        setImageByUri(data.getData());
    }
  }
}
```

撤销和重做功能

Activity 提供绘图操作的撤销和重做功能。

initUndoRedoEnable()
通过设置回调函数初始化撤销和重做功能。

```
void initUndoRedoEnable() {
  drawView.history.setCallBack(new History.CallBack() {
    @Override
    public void onHistoryChanged() {
      setUndoRedoEnable();
      if (curFragmentId != DRAW_FRAGMENT) {
        showDrawFragment(curDrawMode);
      }
    }
```

```
  });
}

void setUndoRedoEnable() {
  redoView.setEnabled(drawView.history.canRedo());
  undoView.setEnabled(drawView.history.canUndo());
}
```

自定义 DrawView

`DrawView` 是一个自定义视图，用于处理绘图操作、触摸事件和缩放。

onTouchEvent(MotionEvent event)
处理绘图和缩放的触摸事件。

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  if (!scaleMode) {
    handleDrawTouchEvent(event);
  } else {
    handleScaleTouchEvent(event);
  }
  return true;
}

private void handleDrawTouchEvent(MotionEvent event) {
  int action = event.getAction();
  float x = event.getX();
  float y = event.getY();
  if (action == MotionEvent.ACTION_DOWN) {
    path.moveTo(x, y);
  } else if (action == MotionEvent.ACTION_MOVE) {
    path.quadTo(preX, preY, x, y);
  } else if (action == MotionEvent.ACTION_UP) {
    Matrix matrix1 = new Matrix();
    matrix.invert(matrix1);
    path.transform(matrix1);
```

```
    paint.setStrokeWidth(strokeWidth * 1.0f / totalRatio);
    history.saveToStack(path, paint);
    cacheCanvas.drawPath(path, paint);
    paint.setStrokeWidth(strokeWidth);
    path.reset();
  }
  setPrev(event);
  invalidate();
}

private void handleScaleTouchEvent(MotionEvent event) {
  switch (event.getActionMasked()) {
    case MotionEvent.ACTION_POINTER_DOWN:
      lastFingerDist = calFingerDistance(event);
      break;
    case MotionEvent.ACTION_MOVE:
      if (event.getPointerCount() == 1) {
        handleMove(event);
      } else if (event.getPointerCount() == 2) {
        handleZoom(event);
      }
      break;
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
      lastMoveX = -1;
      lastMoveY = -1;
      break;
    default:
      break;
  }
}

private void handleMove(MotionEvent event) {
  float moveX = event.getX();
  float moveY = event.getY();
  if (lastMoveX == -1 && lastMoveY == -1) {
```

```
    lastMoveX = moveX;
    lastMoveY = moveY;
  }
  moveDistX = (int) (moveX - lastMoveX);
  moveDistY = (int) (moveY - lastMoveY);
  if (moveDistX + totalTranslateX > 0 || moveDistX + totalTranslateX + curBitmapWidth < width) {
    moveDistX = 0;
  }
  if (moveDistY + totalTranslateY > 0 || moveDistY + totalTranslateY + curBitmapHeight < height) {
    moveDistY = 0;
  }
  status = STATUS_MOVE;
  invalidate();
  lastMoveX = moveX;
  lastMoveY = moveY;
}

private void handleZoom(MotionEvent event) {
  float fingerDist = calFingerDistance(event);
  calFingerCenter(event);
  if (fingerDist > lastFingerDist) {
    status = STATUS_ZOOM_OUT;
  } else {
    status = STATUS_ZOOM_IN;
  }
  scaledRatio = fingerDist * 1.0f / lastFingerDist;
  totalRatio = totalRatio * scaledRatio;
  if (totalRatio < initRatio) {
    totalRatio = initRatio;
  } else if (totalRatio > initRatio * 4) {
    totalRatio = initRatio * 4;
  }
  lastFingerDist = fingerDist;
  invalidate();
}
```

onDraw(Canvas canvas)

绘制视图的当前状态。

```
@Override
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  if (scaleMode) {
    switch (status) {
      case STATUS_MOVE:
        move(canvas);
        break;
      case STATUS_ZOOM_IN:
      case STATUS_ZOOM_OUT:
        zoom(canvas);
        break;
      default:
        if (cacheBm != null) {
          canvas.drawBitmap(cacheBm, matrix, null);
          canvas.drawPath(path, paint);
        }
    }
  } else {
    if (cacheBm != null) {
      canvas.drawBitmap(cacheBm, matrix, null);
      canvas.drawPath(path, paint);
    }
  }
}
```

move(Canvas canvas)
处理缩放期间的移动操作。

```
private void move(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  totalTranslateX = moveDistX + totalTranslateX;
  totalTranslateY = moveDistY + totalTranslateY;
  matrix.postTranslate(totalTranslateX, totalTranslateY);
  canvas.drawBitmap(cacheBm, matrix, null);
```

```
}
```

zoom(Canvas canvas)

处理缩放操作。

```
private void zoom(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  int scaledWidth = (int) (cacheBm.getWidth() * totalRatio);
  int scaledHeight = (int) (cacheBm.getHeight() * totalRatio);
  int translateX;
  int translateY;
  if (curBitmapWidth < width) {
    translateX = (width - scaledWidth) / 2;
  } else {
    translateX = (int) (centerPointX + (totalTranslateX - centerPointX) * scaledRatio);
    if (translateX > 0) {
      translateX = 0;
    } else if (scaledWidth + translateX < width) {
      translateX = width - scaledWidth;
    }
  }
  if (curBitmapHeight < height) {
    translateY = (height - scaledHeight) / 2;
  } else {
    translateY = (int) (centerPointY + (totalTranslateY - centerPointY) * scaledRatio);
    if (translateY > 0) {
      translateY = 0;
    } else if (scaledHeight + translateY < height) {
      translate


Y = height - scaledHeight;
    }
  }
  totalTranslateX = translateX;
  totalTranslateY = translateY;
  curBitmapWidth = scaledWidth;
```

```
  curBitmapHeight = scaledHeight;
  matrix.postTranslate(translateX, translateY);
  canvas.drawBitmap(cacheBm, matrix, null);
}
```

历史管理

`History` 类管理绘图历史记录以实现撤销和重做功能。

saveToStack(Path path, Paint paint)
将当前路径和画笔保存到堆栈中。

```
public void saveToStack(Path path, Paint paint) {
  Draw draw = new Draw();
  draw.path = new Path(path);
  draw.paint = new Paint(paint);
  saveToStack(draw);
}

public void saveToStack(Draw draw) {
  curPos++;
  while (histroy.size() > curPos) {
    histroy.pop();
  }
  histroy.push(draw);
  if (callBack != null) {
    callBack.onHistoryChanged();
  }
}
```

getBitmapAtDraw(int n)
返回表示历史记录中给定点状态的位图。

```
public Bitmap getBitmapAtDraw(int n) {
  Canvas canvas = new Canvas();
  Bitmap bm = Utils.getCopyBitmap(srcBitmap);
  canvas.setBitmap(bm);
  for (int i = 0; i <= n; i++) {
```

```
    Draw draw = histroy.get(i);
    canvas.drawPath(draw.path, draw.paint);
  }
  return bm;
}
```

undo()

执行撤销操作。

```
public Bitmap undo() throws UnsupportedOperationException {
  if (canUndo()) {
    curPos--;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException(" 没有可撤销的记录");
  }
}
```

redo()

执行重做操作。

```
public Bitmap redo() throws UnsupportedOperationException {
  if (canRedo()) {
    curPos++;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException(" 没有可重做的记录");
  }
}
```

canUndo()

检查是否可以撤销。

```
public boolean canUndo() {
```

```
  return curPos > 0;
}
```

canRedo()

检查是否可以重做。

```
public boolean canRedo() {
  return curPos + 1 < histroy.size();
}
```

结论

`DrawActivity` 及其相关类提供了在 Android 中实现自定义绘图视图的全面示例。它展示了各种技术，包括处理触摸事件、管理绘图历史记录以及与其他组件（如 fragment 和异步任务）的集成。通过理解每个组件和算法，您可以在自己的应用中利用这些技术来创建功能强大且互动性强的绘图功能。