深入解析 Android 中的自定義繪圖

本博客文章在 ChatGPT-4o 的幫助下編寫。

介紹

在這篇博客中，我們將探討 `DrawActivity` 類，這是一個在 Android 應用中實現自定義繪圖視圖的全面示例。我們將分解每個組件和使用的算法，詳細解釋它們如何協同工作以實現所需的功能。

目錄

DrawActivity 概述

`DrawActivity` 是處理繪圖操作、圖像裁剪以及與其他組件（如 fragment 和圖像上傳）交互的主要活動。它提供了一個用戶界面，用戶可以在其中繪圖、撤銷、重做和操作圖像。

```java
public class DrawActivity extends Activity implements View.OnClickListener {
  // 請求代碼和 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;

    // 處理圖像和繪圖操作的變量
    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 創建時，執行各種初始化操作，如設置視圖、加載初始圖像和配置事件監聽器。

```
@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 中使用的視圖。

```
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()

設置繪圖視圖的大小。

```
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()

加載將用於繪圖的初始圖像。

```
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);
}
```

處理圖像操作

Activity 處理各種圖像操作，如通過 URI 設置圖像、裁剪和保存繪製的位圖。

setImageByUri(Uri uri)
從給定的 URI 加載圖像並準備繪圖。

```
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)
啟動圖像裁剪活動。

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

saveBitmap()
將繪製的位圖保存到文件並上傳到服務器。

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

saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap)
將位圖保存到文件並異步上傳。

```
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 以處理應用的各種狀態，如繪圖、識別和等待。

showDrawFragment(int infoId)
顯示繪圖 fragment。

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

showWaitFragment()
顯示等待 fragment。

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

showFragment(Fragment fragment)
用指定的 fragment 替換當前 fragment。

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

事件處理

Activity 處理各種用戶交互，如按鈕點擊和菜單選擇。

onClick(View v)
處理不同視圖的點擊事件。

```
@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)
處理其他活動的結果，如圖像選擇或裁剪。

```
@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());
    }
  }
}
```

撤銷和重做功能

Activity 提供繪圖操作的撤銷和重做功能。

initUndoRedoEnable()
通過設置回調函數初始化撤銷和重做功能。

```
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());
}
```

自定義 DrawView

`DrawView` 是一個自定義視圖，用於處理繪圖操作、觸摸事件和縮放。

onTouchEvent(MotionEvent event)
處理繪圖和縮放的觸摸事件。

```
@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;
}
```

```java
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)
繪製視圖的當前狀態。

```java
@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)
處理縮放期間的移動操作。

```
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)
處理縮放操作。

“ ‘java 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) * scaled