Traitement d'images Android simplifié

La manipulation des images dans Android peut être délicate—les bitmaps consomment de la mémoire, les rotations perturbent l'orientation, et le recadrage nécessite de la précision. C'est là que les classes utilitaires comme `BitmapUtils` et `Crop` deviennent pratiques. Dans cet article, je vais vous guider à travers une paire de classes puissantes du package `com.lzw.flower.utils`. Nous explorerons leur code, décomposerons ce que fait chaque méthode et montrerons comment les utiliser dans vos projets. Plongeons-y !

BitmapUtils : Votre Kit d'Outils de Manipulation d'Images de Référence La classe `BitmapUtils` est une collection de méthodes statiques pour manipuler les objets `Bitmap`. Voici le code complet, suivi d'une décomposition :

```
package com.lzw.flower.utils;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.net.Uri;
import android.provider.MediaStore;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class BitmapUtils {
  public static Bitmap convertGreyImg(Bitmap img) {
    int width = img.getWidth();
    int height = img.getHeight();
    int[] pixels = new int[width * height];
    img.getPixels(pixels, 0, width, 0, 0, width, height);
    int alpha = 0xFF << 24;
    for (int i = 0; i < height; i++) {
      for (int j = 0; j < width; j++) {
        int grey = pixels[width * i + j];
        int red = ((grey & 0x00FF0000) >> 16);
        int green = ((grey & 0x0000FF00) >> 8);
        int blue = (grey & 0x000000FF);
```

```
      grey = (int) ((float) red * 0.3 + (float) green * 0.59 + (float) blue * 0.11);
      grey = alpha | (grey << 16) | (grey << 8) | grey;
      pixels[width * i + j] = grey;
    }
  }
  Bitmap result = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
  result.setPixels(pixels, 0, width, 0, 0, width, height);
  return result;
}

public static Bitmap toGreyImg(Bitmap bitmapOrg) {
  Bitmap bitmapNew = bitmapOrg.copy(Bitmap.Config.ARGB_8888, true);
  if (bitmapNew == null) {
    return null;
  }
  for (int i = 0; i < bitmapNew.getWidth(); i++) {
    for (int j = 0; j < bitmapNew.getHeight(); j++) {
      int col = bitmapNew.getPixel(i, j);
      int alpha = col & 0xFF000000;
      int red = (col & 0x00FF0000) >> 16;
      int green = (col & 0x0000FF00) >> 8;
      int blue = (col & 0x000000FF);
      int gray = (int) ((float) red * 0.3 + (float) green * 0.59 + (float) blue * 0.11);
      int newColor = alpha | (gray << 16) | (gray << 8) | gray;
      bitmapNew.setPixel(i, j, newColor);
    }
  }
  return bitmapNew;
}

public static void saveBitmapToPath(Bitmap bitmap, String imagePath) {
  FileOutputStream out = null;
  File file = new File(imagePath);
  if (file.getParentFile().exists() == false) {
    file.getParentFile().mkdirs();
  }
  try {
    out = new FileOutputStream(imagePath);
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, out);
    out.flush();
```

```
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      try {
        if (out != null) out.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }

  public static Bitmap rotateBitmap(Bitmap source, float angle) {
    Matrix matrix = new Matrix();
    matrix.postRotate(angle);
    return Bitmap.createBitmap(source, 0, 0, source.getWidth(), source.getHeight(), matrix, true);
  }

  public static Uri getResourceUri(int resId) {
    return Uri.parse("android.resource://com.lzw.flower/" + resId);
  }

  public static Bitmap getBitmapByUri(Context cxt, Uri uri) throws IOException {
    return MediaStore.Images.Media.getBitmap(cxt.getContentResolver(), uri);
  }

  public static int calInSampleSize(BitmapFactory.Options options, int reqWidth) {
    int w = options.outWidth;
    int h = options.outHeight;
    int inSampleSize = 1;
    if (w > reqWidth && reqWidth > 0) {
      inSampleSize = Math.round(w / reqWidth);
    }
    return inSampleSize;
  }

  public static Bitmap decodeSampledBitmapFromPath(String path, int reqWidth) {
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);
    int inSampleSize = calInSampleSize(options, reqWidth);
```

```
    options.inJustDecodeBounds = false;
    options.inSampleSize = inSampleSize;
    return BitmapFactory.decodeFile(path, options);
  }

  public static Bitmap decodeFileByHeight(String path, int reqH) {
    BitmapFactory.Options opt = new BitmapFactory.Options();
    opt.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, opt);
    int scale = calInSampleSizeByHeight(opt, reqH);
    opt.inSampleSize = scale;
    opt.inJustDecodeBounds = false;
    Bitmap bm = BitmapFactory.decodeFile(path, opt);
    return bm;
  }

  public static int calInSampleSizeByHeight(BitmapFactory.Options options, int reqHeight) {
    int h = options.outHeight;
    int inSampleSize = 1;
    if (h > reqHeight) {
      inSampleSize = Math.round(h * 1.0f / reqHeight);
    }
    return inSampleSize;
  }
}
```

Ce qui est à l'intérieur ?

- **Conversion en Niveaux de Gris** :
  - `convertGreyImg` : Utilise un tableau de pixels pour traiter par lots le bitmap en niveaux de gris.
  - `toGreyImg` : Traite pixel par pixel sur une copie mutable, offrant une approche alternative. Les deux utilisent la formule de luminosité (`0.3R + 0.59G + 0.11B`) pour un niveau de gris naturel.
- **Opérations sur Fichiers** :
  - `saveBitmapToPath` : Enregistre un bitmap en tant que PNG, créant des répertoires si nécessaire.
- **Transformations** :
  - `rotateBitmap` : Fait pivoter une image en utilisant une `Matrix`—simple mais efficace.
- **Chargement et Échantillonnage** :
  - `getBitmapByUri` et `getResourceUri` : Chargent des images à partir d'URIs ou de ressources.

- `decodeSampledBitmapFromPath` et `decodeFileByHeight` : Met à l'échelle efficacement les grandes images par largeur ou hauteur, évitant les problèmes de mémoire.

Crop : Recadrage Précis avec les Outils Natifs d'Android La classe `Crop` utilise l'intention de recadrage intégrée d'Android. Voici le code :

```
package com.lzw.flower.utils;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import com.lzw.flower.base.App;

import java.io.File;

public class Crop {
  public static void startPhotoCrop(Activity cxt, Uri uri, String outputPath, int resultCode) {
    Intent intent = new Intent("com.android.camera.action.CROP");
    intent.setDataAndType(uri, "image/*");
    int w = App.drawWidth;
    int h = App.drawHeight;
    int factor = gcd(w, h);
    int w1 = w / factor;
    int h1 = h / factor;
    intent.putExtra("crop", "true")
        .putExtra("aspectX", w1)
        .putExtra("aspectY", h1)
        .putExtra("scale", true)
        .putExtra("outputX", w)
        .putExtra("outputY", h)
        .putExtra("outputFormat", Bitmap.CompressFormat.PNG.toString());
    intent.putExtra("noFaceDetection", true);
    intent.putExtra("return-data", false);
    Uri uri1 = Uri.fromFile(new File(outputPath));
    intent.putExtra(MediaStore.EXTRA_OUTPUT, uri1);
    cxt.startActivityForResult(intent, resultCode);
```

```
  }

  static int gcd(int a, int b) {
    if (b == 0) {
      return a;
    } else {
      return gcd(b, a % b);
    }
  }
}
```

Ce qui se passe ici ?

- `startPhotoCrop` : Lance l'activité de recadrage système avec un `Uri` spécifié, un rapport d'aspect (simplifié en utilisant le PGCD), et un chemin de sortie. Il suppose que `App.drawWidth` et `App.drawHeight` sont définis ailleurs (par exemple, dans une classe `App` de base).
- `gcd` : Une méthode récursive pour calculer le plus grand commun diviseur, garantissant que le rapport d'aspect est sous sa forme la plus simple.

Tout Assembler : Exemples d'Utilisation Voici comment vous pourriez utiliser ces utilitaires dans une application Android :

```
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import com.lzw.flower.utils.BitmapUtils;
import com.lzw.flower.utils.Crop;

public class MainActivity extends AppCompatActivity {
    private static final int REQUEST_CROP = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Convertir une image en niveaux de gris et l'enregistrer
```

```
        Bitmap original = BitmapFactory.decodeResource(getResources(), R.drawable.sample);
        Bitmap grey = BitmapUtils.convertGreyImg(original);
        BitmapUtils.saveBitmapToPath(grey, "/sdcard/DCIM/grey_image.png");

        // Charger et mettre à l'échelle une image efficacement
        Bitmap scaled = BitmapUtils.decodeSampledBitmapFromPath("/sdcard/DCIM/photo.jpg", 200);

        // Lancer le recadrage
        Uri imageUri = Uri.fromFile(new File("/sdcard/DCIM/photo.jpg"));
        Crop.startPhotoCrop(this, imageUri, "/sdcard/DCIM/cropped.png", REQUEST_CROP);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_CROP && resultCode == RESULT_OK) {
            // L'image recadrée est enregistrée à "/sdcard/DCIM/cropped.png"
        }
    }
}
```

Notes : - Assurez-vous d'avoir les autorisations de stockage appropriées dans votre `AndroidManifest.xml` et des vérifications à l'exécution pour les opérations de fichiers. - La classe `App` (référencée dans `Crop`) doit définir `drawWidth` et `drawHeight`.

Pourquoi Ces Utilitaires Sont Géniaux

1. **Efficacité** : Les méthodes d'échantillonnage empêchent les erreurs `OutOfMemoryError` lors du traitement des grandes images.
2. **Flexibilité** : Les niveaux de gris, la rotation et le recadrage couvrent une large gamme d'utilisations.
3. **Simplicité** : Les méthodes statiques facilitent l'intégration—aucune instantiation requise.

Pensées Finales Les classes `BitmapUtils` et `Crop` sont un excellent point de départ pour toute application Android nécessitant une manipulation d'images. Que vous construisiez un éditeur de photos, optimisiez des vignettes de galerie ou ajoutiez un recadrage piloté par l'utilisateur, ce code vous couvre. Essayez-le, ajustez-le à vos besoins et faites-moi savoir comment cela fonctionne pour vous !

Quels défis de traitement d’images avez-vous rencontrés dans Android ? Partagez vos pensées ci-dessous !