Лабораторна робота №13. Робота з зображеннями Bitmap

Мета: вивчити порядок роботи з зображеннями Bitmap в Android великого розміру.

Теоретичні відомості

Загальні відомості про процеси та потоки. Під **процесом** зазвичай розуміють окремо запущену програму. За замовчуванням всі компоненти одного додатку працюють в одному процесі і потоці (який називається головним потоком). Якщо компонент додатку запускається за наявності процеса для цього додатку, тоді цей компонент відноситься до того ж поцеса і потока.

Однак, можна організувати виконання різних компонентів додатку в окремих процесах та створювати окремі потоки.

Якщо необхідно контролювати до якого процеса відноситься певний компонент додатка, це можна зробити у файлі маніфеста додатка. Для цього є атрибут **android:process**, який дозволяє задавати процес, у якому слід виконувати певний компонент (<activity>, <service>, <receiver> чи <provider>)

Елемент **<application>** також має атрибут **android:process** який дозволяє задавати значення за замовчуванням для всіх компонентів додатка.

При примусовому припиненні дії процесів у разі недостачі оперативної пам'яті враховується пріоритет процеса (найбільш важливий для користувача процес видаляється останнім). Існує п'ять рівнів пріоритету.

- **Процес переднього плана**. Процес, необхідний для поточної діяльності користувача. Процес вважається процесом переднього плана, якщо виконуються такі умови:
  1. Містить activity, з яким взаємодіє користувач (викликано метод onResume()).
- **Видимий процес**. Процеси, що не містять компонентів переднього плана, але можуть впливати на відображення на екрані.
  1. Містить activity, що не знаходиться не передньому плані, але видно для користувача (викликано метод onPause()).
- **Службовий процес**. Процес, який виконує системну службу та не підпадає під перші дві категорії
- **Фоновий процес**. Процес, який виконує певні дії, які не видимі користувачу, та не впливають на поведінку користувача.
- **Порожній процес**. Який не містить ніяких компонент активного додатку.

При запуску додатку система створює **потік** виконання додатку (який називається головним). Інша назва головного потоку – потік користувацького інтерфейса (UI), оскільки в ньому система взаємодіє з компонентами UI.

Оскільки всі компоненти додатка використовують головний потік, при ресурсоємних операціях (мережевий доступ, запит до бази даних) додаток може блокувати інтерфейс користувача.

Для ефективної роботи з основним потоком використовують такі правила:

1. Не блокувати UI потік.
2. Не звертатися до віджетів та інших компонент інтерфейсу з за меж UI потока.

Робочі потоки. Таким чином, важливо не виконувати «важких» операцій в UI-потоці. Такі операції виконуються в окремих потоках («фонових» або «робочих» потоках).

Приклад 1. Метод onClick(), який відображає зображення з окремого потока та відображає його у віджеті ImageView (рис. 1).

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Рис. 1

Приклад 1, однак, порушує друге правило ефективної роботи з основним потоком: не звертатися до віджетів та інших компонент інтерфейсу з за меж UI потока. Це може привести до непередбаченої поведінки додатку.

Для усунення цієї проблеми Android пропонує декілька методів доступу до користувацького інтерфейса з інших потоків:

- **Activity.runOnUiThread(Runnable).**
- **View.post(Runnable).**
- **View.postDelayed(Runnable, long).**

Приклад 2. Аналогічна задача до прикладу 1, але використовується метод **View.post(Runnable)**:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Рис. 2

Використання AsyncTask. Методи класу **AsyncTask** дозволяють публікувати в UI потоці результати з робочого потоку. При цьому, немає необхідності самостійно оброблювати потоки.

Необхідно створити підклас AsyncTask і реалізувати метод зворотного виклику **doInBackground()**, який працює у фоновому потоці.

Для оновлення користувацького інтерфейсу реалізується метод **onPostExecute()**, який доставляє результати з метода **doInBackground()** і працює в UI потоці.

Задача виконується через виклик методу **execute()** з користувацького інтерфейсу.

Приклад 3.

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Рис. 3

Завантаження великих зображень Bitmap. Зазвичай, при відображенні зображень з великої роздільною здатністю, в пам’ять завантажується зображення з меншою роздільною здатністю і потім відображається у відповідному віджеті.

Клас **BitmapFactory** містить декілька методів для декодування зображень (**decodeByteArray()**, **decodeFile()**, **decodeResource()**, і т.д.) з різних джерел. Кожний з методів декодування потребує визначення деякої кількості властивостей за допомогою класа **BitmapFactory.Options** (рис. 4).

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), R.id.myimage, options);
int imageHeight = options.outHeight;
int imageWidth = options.outWidth;
String imageType = options.outMimeType;
```

Рис. 4

Якщо визначені параметри розміру зображень, вони можуть бути використані для визначення способу відображення зображення. Тобто, чи може зображення повністю бути завантажене в пам’ять, чи потрібно згенерувати і відображати меншу копію зображення.

Наприклад, не варто завантажувати зображення розміром 1024x768 пікселя, якщо воно буде відображатися у віджеті ImageView розміром 128x96 пікселя.

Наприклад, для того, щоб створити зменшену копію зображення за допомогою декодера, слід встановити опцію **inSampleSize** в значення true в відповідному об’єкті **BitmapFactory.Options**.

Метод, підрахування розміру зменшеної копії зображення може виглядати наступним чином:

```
public static int calculateInSampleSize(
            BitmapFactory.Options options, int reqWidth, int reqHeight) {
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {

        final int halfHeight = height / 2;
        final int halfWidth = width / 2;

        // Calculate the largest inSampleSize value that is a power of 2 and keeps both
        // height and width larger than the requested height and width.
        while ((halfHeight / inSampleSize) >= reqHeight
                && (halfWidth / inSampleSize) >= reqWidth) {
            inSampleSize *= 2;
        }
    }

    return inSampleSize;
}
```

Рис. 5

А відповідний метод декодування з використанням нових розмірів зображення виглядає так:

```
public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
        int reqWidth, int reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(res, resId, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeResource(res, resId, options);
}
```

Рис. 6

Тоді, для завантаження зменшеної копії зображення у **ImageView** потрібно виконати метод **decodeSampledBitmapFromResource**:

```
mImageView.setImageBitmap(
    decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 100));
```

Рис. 7

Зазвичай, декодування та завантаження зображень за допомогою методів з класа **BitmapFactory.decode** виконується у робочому потоці з використанням AsyncTask, щоб не блокувати головний UI поток.

Приклад 4. Завантаження великого зображення у ImageView з використанням **AsyncTask** та методу **decodeSampledBitmapFromResource()**:

```
class BitmapWorkerTask extends AsyncTask<Integer, Void, Bitmap> {
    private final WeakReference<ImageView> imageViewReference;
    private int data = 0;

    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage collected
        imageViewReference = new WeakReference<ImageView>(imageView);
    }

    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        data = params[0];
        return decodeSampledBitmapFromResource(getResources(), data, 100, 100));
    }

    // Once complete, see if ImageView is still around and set bitmap.
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
...

public void loadBitmap(int resId, ImageView imageView) {
        BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        task.execute(resId);
        }
```

Завдання до лабораторної роботи

1. Створити нове activity з елементом ImageView та кнопкою для вибору графічного файла для відображення. Реалізувати метод для перекодування у зображення меншого розміру та використати AsyncTask для вивід зображення у ImageView

Контрольні запитання

1. Дайте визначення потока та процеса.
2. Які існують типи потоків та процесів?
3. Принципи роботи з UI потоком.