

Отримання інформації про піксельні розміри зображення та його тип

Детальніше: <http://developer.android.com/training/displaying-bitmaps/load-bitmap.html>

Клас [BitmapFactory](#) надає декілька методів декодування ([decodeByteArray\(\)](#), [decodeFile\(\)](#), [decodeResource\(\)](#), etc.) для створення [Bitmap](#) з різних джерел. Обирайте метод залежно від джерела зображення.

Ці методи здійснюють спробу виділити пам'ять для об'єкта bitmap, що створюється, тому легко можуть викликати виняткову ситуацію OutOfMemory.

Кожен із цих методів має додатковий ключ, котрий дозволяє вказати параметри декодування за допомогою класу [BitmapFactory.Options](#). Встановлення значення властивості [inJustDecodeBounds](#) = true при декодуванні перешкоджає виділенню пам'яті; метод повертає null для об'єкта bitmap, проте встановлює [outWidth](#), [outHeight](#) та [outMimeType](#). Такий підхід дозволяє прочитати розміри та тип зображення перед створенням (виділення пам'яті) bitmap.

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
File storageDir = Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES);
File file = new File(storageDir + File.separator + "test.jpg");
Bitmap bMap =
    BitmapFactory.decodeFile(file.getPath(), options);
int imageHeight = options.outHeight;
int imageWidth = options.outWidth;
String imageType = options.outMimeType;
Log.i("image_info", imageWidth + " " + imageHeight + " " + imageType);
```

Результат (Nexus 5): I/image_info: 2368 3200 image/jpeg

Завантаження зменшеної версії у пам'ять

Детальніше: <http://developer.android.com/training/displaying-bitmaps/load-bitmap.html>

Маючи розміри зображення, можна робити висновок про потребу масштабування (зменшення) його чи завантаження у пам'ять повнорозмірної версії. Деякі фактори, що впливають на прийняття рішення:

- прогнозоване використання пам'яті при завантаженні повнорозмірного зображення у пам'ять;
- об'єм пам'яті, котрий може бути виділений для завантаження зображення у пам'ять;
- розміри цільового ImageView чи іншого елемента інтерфейсу користувача, у котрий повинно бути завантажено зображення;
- розмір екрану пристрою та його роздільна здатність.

Для того, щоб декодер зменшив зображення, встановіть [inSampleSize](#) = true в об'єкті [BitmapFactory.Options](#).

Нижче наведено метод для обчислення розміру коефіцієнта масштабування, котрий є степенем двійки та базується на цільовій ширині та висоті.

```

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;
}

```

Степінь двійки обчислюють тому, що декодер використовує значення, округлене у біт зменшення до найближчого степеня двійки ([inSampleSize](#)). Для того, щоб скористатися цим методом, слід спочатку декодувати з [inJustDecodeBounds](#) = true, передаючи options як параметр, а потім декодувати ще раз з використанням нового значення [inSampleSize](#) та встановивши [inJustDecodeBounds](#) = false.

```

public static Bitmap decodeSampledBitmapFromFile(String path,
    int reqWidth, int reqHeight) {
    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);
    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeFile(path, options);
}

```

Цей метод дозволяє завантажувати зображення довільного розміру у [ImageView](#), що відображає мініатюру певного розміру, наприклад, 100x100:

```

Bitmap bMap = decodeSampledBitmapFromFile(file.getPath(), 100, 100);
mImageView.setImageBitmap(bMap);

```

Аналогічним чином можна декодувати растрові зображення з інших джерел, підставляючи відповідні [BitmapFactory.decode*](#) методи.

Обробка растрових зображень поза UI-потокom

Методи [BitmapFactory.decode*](#), розглянуті у [Load Large Bitmaps Efficiently](#), не слід виконувати у потоковій UI, якщо джерело даних — мережа чи пристрій зберігання (або будь-яке джерело окрім оперативної пам'яті). Час, потрібний на завантаження таких даних, не є передбачуваним і залежить від значної кількості факторів (швидкості читання з мережі та пристрою зберігання, розміру зображення, потужності CPU тощо). Якщо одна із таких задач блокує виконання потоку інтерфейсу користувача UI, операційна система позначає таку програму як такою, що не відповідає на зовнішні запити, і надає користувачеві можливість завершити її виконання (детальніше: [Designing for Responsiveness](#)).

Нижче розглянуто обробку растрових зображень у фоновому потоковій за допомогою класу [AsyncTask](#) та показано, як уникнути проблем, пов'язаних із конкуренцією потоків.

Джерело:

```
class BitmapWorkerTask extends AsyncTask<String, Void, Bitmap> {
    private final WeakReference<ImageView> imageViewReference;
    private String data = null;
    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage collected
        imageViewReference = new WeakReference<>(imageView);
    }
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(String... params) {
        data = params[0];
        return decodeSampledBitmapFromFile(data, 512, 512);
    }
    // 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);
            }
        }
    }
}
```

`imageViewReference` — слабке посилання на `imageView`. Якщо з деяких причин (наприклад, зміни `activity`) `garbage collector` повинен звільнити пам'ять, виділену для `imageView`, сильне посилання на об'єкт `imageView` завадить це зробити. Слабке посилання не заважає роботі `garbage collector`.

Параметри `AsyncTask<String, Void, Bitmap>` означають: `String` — тип параметра, котрий передають у `doInBackground()`, `Void` — під час роботи потоку відсутні виклики `publishProgress()`, `Bitmap` — тип `doInBackground()`, його ж передають у `onPostExecute()`.

Виконання потоку починають за допомогою методу `execute()`:

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

Параметр `execute()` отримує метод `doInBackground(String... params)` потоку. Доступ до адреси файлу, що містить растрове зображення, здійснюють за допомогою `data = params[0]` (трикрапка позначає `params` як `vararg`). Після цього запущений потік декодує растрове зображення із зовнішнього файлу (у даному випадкові — із зовнішнього пристрою) `decodeSampledBitmapFromFile(data, 512, 512)`. Результатом виконання є `Bitmap`.

У `onPostExecute()` слід перевірити, чи існує `imageView`, з яким пов'язано слабке посилання `imageViewReference`:

```
if (imageViewReference != null && bitmap != null) {
```

Посилання `imageViewReference` не дорівнюватиме `null`, якщо на момент його ініціалізації існував об'єкт `imageView`. Відповідно, `bitmap` не дорівнюватиме `null`, якщо декодування растрового зображення було успішним.

Виконання `imageViewReference.get()` повертає посилання на об'єкт `imageView`. Якщо на даний момент такий об'єкт ще існує, для нього встановлюють `ImageBitmap`:

```
imageView.setImageBitmap(bitmap);
```

Повний текст класу `Activity` (змінить ім'я пакету у відповідності до власного проекту):

```
package com.example.student.asyncTask;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.os.AsyncTask;  
import android.os.Environment;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.ImageView;  
import java.io.File;  
import java.lang.ref.WeakReference;  
public class MainActivity extends AppCompatActivity {  
    ImageView mImageView;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
mImageView = (ImageView) findViewById(R.id.image);
File storageDir = Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES);
File file = new File(storageDir + File.separator + "test.jpg");
loadBitmap(file.getPath(), mImageView);
}

class BitmapWorkerTask extends AsyncTask<String, Void, Bitmap> {
    private final WeakReference<ImageView> imageViewReference;
    private String data = null;
    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage collected
        imageViewReference = new WeakReference<>(imageView);
    }
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(String... params) {
        data = params[0];
        return decodeSampledBitmapFromFile(data, 512, 512);
    }
    // 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(String path, ImageView imageView) {
    BitmapWorkerTask task = new BitmapWorkerTask(imageView);
    task.execute(path);
}

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;
}
public static Bitmap decodeSampledBitmapFromFile(String path,
                                                int reqWidth, int reqHeight) {
    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);
    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeFile(path, options);
}
}

```

Layout:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.vitaliy.cameraintent.MainActivity">

    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>

```

Конкуренція потоків