Класс BitmapFactory

Класс **BitmapFactory** позволяет создать объект **Bitmap** из файла, потока или байтового массива. Данный класс часто используется в работе с изображениями.

Класс имеет несколько методов **decode\*** (**decodeByteArray(), decodeFile(), decodeResource()** и др.) для создания растрового изображения из различных источников. Не забывайте, что методы пытаются выделить память под создаваемое изображение и вы можете получить ошибку из-за нехватки памяти на устройстве. Обратите внимание, что каждый метод имеет вторую перегруженную версию с тем же именем и дополнительным параметром типа **BitmapFactory.Options**.

Мы можем загрузить изображение из каталога **assets**:

InputStream inputStream = assetManager.open("cat.png");

Bitmap bitmap = BitmapFactory.decodeStream(inputStream);

Класс **BitmapFactory.Options** имеет несколько полезных свойств, которые нужно использовать в своих приложениях.

* inJustDecodeBounds
* inSampleSize
* inBitmap
* inPreferredConfig
* inDensity
* inDither
* inPurgeable

inJustDecodeBounds

Используя свойство **inJustDecodeBounds** со значением *true*, вы можете избежать выделения памяти под изображение, при этом вы можете получить значения ширины (**outWidth**), высоты (**outHeight**) и MIME-типа (**outMimeType**). Подобный приём позволяет прочитать размеры картинки и использовать их для своих целей, сведя к минимуму потребление памяти.

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;

Когда размеры изображения известны, вы уже можете решать - загружать ли полное изображение или сделать уменьшенную копию. Тут зависит от логики вашего приложения. Если это просмотр фотографий, то нужно следить за потреблением памяти. Если картинка загружается в маленький **ImageView**, то нет смысла использовать настоящее большое изображение. Когда вы определились, что делать дальше, то установите снова значение **false** для данного свойства.

inSampleSize

Чтобы декодер пережал картинку, установите **inSampleSize** в нужное значение в объекте **BitmapFactory.Options**. Например, если изображение с размерами 2048x1536 сжать до размеров 512x384 (в 4 раза), то изображение в памяти будет занимать 0.75мб вместо 12мб. Разница ощутима. Использовать нужно значения, кратные двум. Если вы по ошибке установите другое значение, то оно всё равно будет уменьшено до блишайшего меньшего значения, кратное двум.

Ниже приводится метод для вычисления новых размеров изображения по заданными ширине и высоте, чтобы изменение размера было пропорциональным.

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) {

// Calculate ratios of height and width to requested height and width

final int heightRatio = Math.round((float) height / (float) reqHeight);

final int widthRatio = Math.round((float) width / (float) reqWidth);

// Choose the smallest ratio as inSampleSize value, this will guarantee

// a final image with both dimensions larger than or equal to the

// requested height and width.

inSampleSize = heightRatio < widthRatio ? heightRatio : widthRatio;

}

return inSampleSize;

}

Чтобы использовать этот метод, сначала декодируйте изображение через **inJustDecodeBounds = true**, затем декодируйте снова, используя новые значения **inSampleSize** и **inJustDecodeBounds = false**:

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);

}

Чтобы загрузить большое изображение в ImageView с размером 100х100 пикселей, используйте наш метод:

mImageView.setImageBitmap(

decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 100));

По такому же принципу вы можете написать сопутствующие методы для других декодирующих методов класса **BitmapFactory**.

Сама операция сжатия картинки очень трудоёмка и её не стоит делать в основном потоке приложения. Лучше её вынести в отдельный поток через **AsyncTask**.

Примеры есть в документации: [Processing Bitmaps Off the UI Thread](http://developer.android.com/intl/ru/training/displaying-bitmaps/process-bitmap.html)

Другие свойства используются реже. Поэтому информация только в ознакомительных целях.

inBitmap

Если передать в этот параметр объект **Bitmap**, то он и будет использован для получения итогового результата вместо создания нового объекта. В Android 4.4 (API 19) передаваемый **Bitmap** должен быть не меньше по размеру (в байтах), чем читаемое изображение. В старых версиях объект должен быть строго того же размера (ширина/высота), что и читаемое изображение. Также необходимо использовать свойство **inSampleSize = 1**.

inPreferredConfig

Настраивается конфигурация [Bitmap.Config](http://developer.alexanderklimov.ru/android/catshop/bitmap.php" \l "config).

inDensity

Задаёт плотность для объекта **Bitmap**.

inDither

Сглаживание цветовой палитры. Где-то валялся пример. Если найду, то добавлю.

inPurgeable

При проблемах с памятью разрешаем системе временно удалить объект **Bitmap**. Когда потребуется вывести картинку на экран, то объект восстанавливается. Естественно, при этом падает производительность из-за повторной работы по обработке изображения.

Об остальных свойствах читайте в документации.