Класс Bitmap

[Bitmap.Config](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#config)  
[Получить Bitmap из ImageView](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#getbitmapiv)  
[Изменение размеров - метод createScaledBitmap()](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#scalebitmap)  
[Кадрирование - метод createBitmap()](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#cadri)  
[Меняем цвета каждого пикселя](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#swapcolor)  
[Конвертируем Bitmap в байтовый массив и обратно](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#bitmaparray)  
[Как раскодировать Bitmap из Base64](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#base64)  
[Вычисляем средние значения цветов](http://developer.alexanderklimov.ru/android/catshop/bitmap.php#average)

Вам часто придётся иметь дело с изображениями котов, которые хранятся в файлах JPG, PNG, GIF. По сути, любое изображение, которое мы загружаем из графического файла, является набором цветных точек (пикселей). А информацию о каждой точке можно сохранить в битах. Отсюда и название - карта битов или по-буржуйски - **bitmap**. У нас иногда используется термин *растр* или растровое изображение. В Android есть специальный класс **android.graphics.Bitmap** для работы с подобными картинками.

Существуют готовые растровые изображения в файлах, о которых поговорим ниже. А чтобы создать с нуля объект **Bitmap** программным способом, нужно вызвать метод **createBitmap()**:

Bitmap bitmap = Bitmap.createBitmap(100, 100,

Bitmap.Config.ARGB\_8888);

В результате получится прямоугольник с заданными размерами в пикселях (первые два параметра). Третий параметр отвечает за информацию о прозрачности и качестве цвета (в конце статьи есть примеры).

Очень часто нужно знать размеры изображения. Чтобы узнать его ширину и высоту в пикселах, используйте соответствующие методы:

int width = bitmap.getWidth();

int height = bitmap.getHeight();

Bitmap.Config

Кроме размеров, желательно знать цветовую схему. У класса **Bitmap** есть метод **getConfig()**, который возвращает перечисление **Bitmap.Config**.

Всего существует несколько элементов перечисления.

* **Bitmap.Config ALPHA\_8** - каждый пиксель содержит в себе информацию только о прозрачности, о цвете здесь ничего нет. Каждый пиксель требует 8 бит (1 байт) памяти.
* **Bitmap.Config ARGB\_4444** - устаревшая конфигурация, начиная с API 13. Аналог ARGB\_8888, только каждому ARGB-компоненту отведено не по 8, а по 4 бита. Соответственно пиксель весит 16 бит (2 байта). Рекомендуется использовать **ARGB\_8888**
* **Bitmap.Config ARGB\_8888** - на каждый из 4-х ARGB-компонентов пикселя (альфа, красный, зеленый, голубой) выделяется по 8 бит (1 байт). Каждый пиксель занимает 4 байта. Обладает наивысшим качеством для картинки.
* **Bitmap.Config RGB\_565** - красному и и синему компоненту выделено по 5 бит (32 различных значений), а зелёному - шесть бит (64 возможных значений). Картинка с такой конфигурацией может иметь артефакты. Каждый пиксель будет занимать 16 бит или 2 байта. Конфигурация не хранит информацию о прозрачности. Можно использовать в тех случаях, когда рисунки не требуют прозрачности и высокого качества.

Конфигурация **RGB\_565** была очень популярна на старых устройствах. С увеличением памяти и мощности процессоров данная конфигурация теряет актуальность.

Разница между качеством картинки показана в статье [Bitmap quality, banding and dithering](http://www.curious-creature.org/2010/12/08/bitmap-quality-banding-and-dithering/).



В большинстве случаев вы можете использовать **ARGB\_8888**.

Получив объект в своё распоряжение, вы можете управлять каждой его точкой. Например, закрасить его синим цветом.

public void onClick(View view) {

ImageView imageView = (ImageView) findViewById(R.id.imageView);

Bitmap bitmap = Bitmap.createBitmap(100, 100,

Bitmap.Config.ARGB\_8888);

// Закрашиваем синим цветом

bitmap.eraseColor(Color.BLUE);

// Выведем результат в ImageView для просмотра

imageView.setImageBitmap(bitmap);

}

Чтобы закрасить отдельную точку, используйте метод **setPixel()** (парный ему метод **getPixel** позволит узнать информацию о точке). Закрасим красной точкой центр синего прямоугольника из предыдущего примера - имитация следа от лазерной указки. Котам понравится.

bitmap.eraseColor(Color.BLUE);

bitmap.setPixel(50, 50, Color.RED);

В нашем случае мы создали растровое изображение самостоятельно и можем на него воздействовать. Но если вы загрузите готовое изображение из файла и попытаетесь добавить к нему красную точку, то можете получить крах программы. Изображение может быть неизменяемым, что-то типа "Только для чтения", помните об этом.

Созданный нами цветной прямоугольник и управление отдельными точками не позволят вам нарисовать фигуру, не говоря уже о полноценном рисунке. Класс **Bitmap** не имеет своих методов для рисования, для этого есть метод **Canvas** (Холст), на котором вы можете размещать объекты **Bitmap**.

}

Когда вы размещали в разметке активности компонент **ImageView** и присваивали атрибуту **android:src** ресурс из папок **drawable-xxx**, то система автоматически выводила изображение на экран.

Если нужно программно получить доступ к битовой карте (изображению) из ресурса, то используется такой код:

// Конвертируем Drawable в Bitmap

Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.catpic);

int mPhotoWidth = bitmap.getWidth();

int mPhotoHeight = bitmap.getHeight();

// присваиваем ImageView

imageView.setImageBitmap(bitmap);

Обратный процес конвертации из **Bitmap** в **Drawable**:

Drawable drawable = new BitmapDrawable(bitmap); // устаревший конструктор

Drawable drawable = new BitmapDrawable(getResources(), bitmap); // новый конструктор

Изображение можно сохранить, например, на SD-карту в виде файла (кусок кода):

try {

FileOutputStream fos = new FileOutputStream(dirname + "cat.jpg");

bitmap.compress(CompressFormat.JPEG, 75, fos);

fos.flush();

fos.close();

} catch (Exception e) {

Log.e("MyLog", e.toString());

}

Каждая точка изображения представлена в виде 4-байтного целого числа. Сначала идёт байт прозрачности - значение 0 соответствует полной прозрачности, а 255 говорит о полной непрозрачности. Промежуточные значения позволяют делать полупрозрачные изображения. Этим искусством в совершенстве владел чеширский кот, который умело управлял всеми точками своего тела и растворялся в пространстве, только улыбка кота долго ещё висела в воздухе (что-то я отвлёкся).

Следующие три байта отвечают за красный, зелёный и синий цвет, которые работают по такому же принципу. Т.е. значение 255 соответствует насыщенному красному цвету и т.д.

Так как любое изображение кота - это набор точек, то с помощью метода **getPixels()** мы можем получить массив этих точек, сделать с этой точкой что-нибудь нехорошее (поменять прозрачность или цвет), а потом с помощью родственного метода **setPixels()** записать новые данные обратно в изображение. Так можно перекрасить чёрного кота в белого и наоборот. Если вам нужна конкретная точка на изображении, то используйте методы **getPixel()/setPixel()**. Подобный подход используется во многих графических фильтрах. Учтите, что операция по замене каждой точки в большом изображении занимает много времени. Желательно проводить подобные операции в отдельном потоке.

На этом базовая часть знакомства с битовой картой закончена. Теперь подробнее.

Учитывая ограниченные возможности памяти у мобильных устройств, следует быть осторожным при использовании объекта **Bitmap** во избежание утечки памяти. Не забывайте освобождать ресурсы при помощи метода **recycle()**, если вы в них не нуждаетесь. Например:

Bitmap image = ..... (ваш код)

...

if(image != null) {

image.recycle();

image = null;

}

Почему это важно? Если не задумываться о ресурсах памяти, то можете получить ошибку **OutOfMemoryError**. На каждое приложение выделяется ограниченное количество памяти (heap size), разное в зависимости от устройства. Например, 16мб, 24мб и выше. Современные устройства как правило имеют 24мб и выше, однако это не так много, если ваше приложение злоупотребляет графическими файлами.

**Bitmap** на каждый пиксель тратит в общем случае 2 или 4 байта (зависит от битности изображения – 16 бит RGB\_555 или 32 бита ARGB\_888). Можно посчитать, сколько тратится ресурсов на **Bitmap**, содержащий изображение, снятое на 5-мегапиксельную камеру.

При соотношении сторон 4:3 получится изображение со сторонами 2583 х 1936. В конфигурации **RGB\_555** объект **Bitmap** займёт 2592 \* 1936 \* 2 = около 10Мб, а в **ARGB\_888** (режим по умолчанию) в 2 раза больше – чуть более 19Мб.

Во избежание проблем с памятью прибегают к помощи методов **decodeXXX()** класса [BitmapFactory](http://developer.alexanderklimov.ru/android/catshop/bitmapfactory.php).

Если установить атрибут **largeHeap** в манифесте, то приложению будет выделен дополнительный блок памяти.

Ещё одна потенциальная проблема. У вас есть **Bitmap** и присвоили данный объект кому-то. Затем объект был удалён из памяти, а ссылка на него осталась. Получите крах приложения с ошибкой типа "Exception on Bitmap, throwIfRecycled".

Возможно, лучше сделать копию.

Bitmap src = // Ваш источник

Bitmap newBitmap = src.copy(src.getConfig(), src.isMutable() ? true : false);

Получить Bitmap из ImageView

Если в **ImageView** имеется изображение, то получить **Bitmap** можно следующим образом:

Bitmap bitmap = ((BitmapDrawable)image.getDrawable()).getBitmap();

Но с этим способом нужно быть осторожным. Например, если в **ImageView** используются элементы **LayerDrawable**, то возникнет ошибка. Можно попробовать такой вариант.

Bitmap bitmap = ((BitmapDrawable)((LayerDrawable)image.getDrawable()).getDrawable(0)).getBitmap(‌​);

Более сложный вариант, но и более надёжный.

imageView.setDrawingCacheEnabled(true);

imageView.measure(MeasureSpec.makeMeasureSpec(0, MeasureSpec.UNSPECIFIED),

MeasureSpec.makeMeasureSpec(0, MeasureSpec.UNSPECIFIED));

imageView.layout(0, 0,

imageView.getMeasuredWidth(), imageView.getMeasuredHeight());

imageView.buildDrawingCache(true);

Bitmap bitmap = Bitmap.createBitmap(imageView.getDrawingCache());

imageView.setDrawingCacheEnabled(false);

Изменение размеров - метод createScaledBitmap()

С помощью метода **createScaledBitmap()** можно изменить размер изображения.

Будем тренироваться на кошках. Добавим картинку в ресурсы (res/drawable). В разметку добавим два элемента **ImageView**

ImageView firstImageView = (ImageView) findViewById(R.id.imageView1);

ImageView secondImageView = (ImageView) findViewById(R.id.imageView2);

// Конвертируем Drawable в Bitmap и выводим в ImageView

Bitmap bmOriginal = BitmapFactory.decodeResource(getResources(),

R.drawable.catlove);

firstImageView.setImageBitmap(bmOriginal);

// Вычисляем ширину и высоту изображения

int width = bmOriginal.getWidth();

int height = bmOriginal.getHeight();

// Половинки

int halfWidth = width / 2;

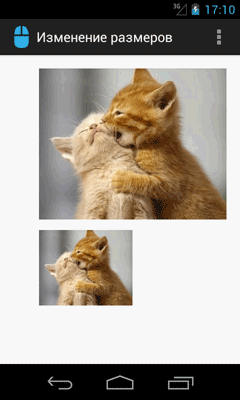
int halfHeight = height / 2;

// Выводим уменьшенную в два раза картинку во втором ImageView

Bitmap bmHalf = Bitmap.createScaledBitmap(bmOriginal, halfWidth,

halfHeight, false);

secondImageView.setImageBitmap(bmHalf);



В последнем параметре у метода идёт булева переменная, отвечающая за сглаживание пикселей. Обычно его применяют, когда маленькое изображение увеличивают в размерах, чтобы улучшить качество картинки. При уменьшении, как правило, в этом нет такой необходимости.

Кадрирование - метод createBitmap()

Существует несколько перегруженных версий метода **Bitmap.createBitmap()**, с помощью которых можно скопировать участок изображения.

* **сreateBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)** - Returns an immutable bitmap from subset of the source bitmap, transformed by the optional matrix.
* **createBitmap(int width, int height, Bitmap.Config config)** - Returns a mutable bitmap with the specified width and height.
* **createBitmap(Bitmap source, int x, int y, int width, int height)** - Returns an immutable bitmap from the specified subset of the source bitmap.
* **createBitmap(int[] colors, int offset, int stride, int width, int height, Bitmap.Config config)** - Returns a immutable bitmap with the specified width and height, with each pixel value set to the corresponding value in the colors array.
* **createBitmap(Bitmap src)** - Returns an immutable bitmap from the source bitmap.
* **createBitmap(int[] colors, int width, int height, Bitmap.Config config)** - Returns a immutable bitmap with the specified width and height, with each pixel value set to the corresponding value in the colors array.

Описываемый ниже код не является оптимальным и очень ресурсоёмкий. На больших изображениях код будет сильно тормозить. Приводится для ознакомления. Чтобы вывести часть картинки, можно сначала создать нужный Bitmap с заданными размерами, занести в массив каждый пиксель исходного изображения, а затем этот же массив вернуть обратно. Но, так как мы уже задали другие размеры, то часть пикселей не выведутся.

ImageView image1 = (ImageView) findViewById(R.id.imageView1);

ImageView image2 = (ImageView) findViewById(R.id.imageView2);

// Конвертируем Drawable в Bitmap и выводим в ImageView

Bitmap bmOriginal = BitmapFactory.decodeResource(getResources(),

R.drawable.catlove);

image1.setImageBitmap(bmOriginal);

// Вычисляем ширину и высоту изображения

int width = bmOriginal.getWidth();

int height = bmOriginal.getHeight();

// Половинки

int halfWidth = width / 2;

int halfHeight = height / 2;

// Выводим верхнюю левую четвертинку картинки

Bitmap bmUpRightPartial = Bitmap.createBitmap(halfWidth, halfHeight,

Bitmap.Config.ARGB\_8888);

int[] pixels = new int[halfWidth \* halfHeight];

bmOriginal

.getPixels(pixels, 0, halfWidth, 0, 0, halfWidth, halfHeight);

bmUpRightPartial

.setPixels(pixels, 0, halfWidth, 0, 0, halfWidth, halfHeight);

image2.setImageBitmap(bmUpRightPartial);

По аналогии мы можем вывести и нижнюю правую часть изображения:

// Выводим нижнюю правую четвертинку

Bitmap bmDownRightPartial = Bitmap.createBitmap(halfWidth, halfHeight,

Bitmap.Config.ARGB\_8888);

int[] pixels5 = new int[halfWidth \* halfHeight];

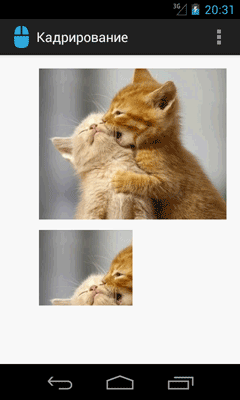
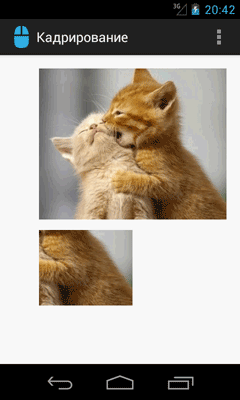
bmOriginal.getPixels(pixels5, 0, halfWidth, halfWidth, halfHeight,

halfWidth, halfHeight);

bmDownRightPartial

.setPixels(pixels5, 0, halfWidth, 0, 0, halfWidth, halfHeight);

image2.setImageBitmap(bmDownRightPartial);

Немного модифицировав код, мы можем кадрировать центр исходного изображения. Предварительно придётся проделать несколько несложных вычислений.

// Центр 1/4

Bitmap bmCenterPartial = Bitmap.createBitmap(halfWidth, halfHeight,

Bitmap.Config.ARGB\_8888);

int[] pixels = new int[halfWidth \* halfHeight];

bmOriginal.getPixels(pixels, 0, halfWidth, halfWidth / 2,

halfHeight / 2, halfWidth, halfHeight);

bmCenterPartial

.setPixels(pixels, 0, halfWidth, 0, 0, halfWidth, halfHeight);

image2.setImageBitmap(bmCenterPartial);

Скриншот приводить не буду, проверьте самостоятельно.

Меняем цвета каждого пикселя

Через метод **getPixels()** мы можем получить массив всех пикселей растра, а затем в цикле заменить определённым образом цвета в пикселе и получить перекрашенную картинку. Для примера возьмем стандартный значок приложения, поместим его в ImageView, извлечём информацию из значка при помощи метода **decodeResource()**, применим собственные методы замены цвета и полученный результат поместим в другие ImageView:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<ImageView

android:id="@+id/image1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<ImageView

android:id="@+id/image2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<ImageView

android:id="@+id/image3"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<ImageView

android:id="@+id/image4"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

</LinearLayout>

Код для класса активности:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_test);

ImageView image1 = (ImageView)findViewById(R.id.image1);

ImageView image2 = (ImageView)findViewById(R.id.image2);

ImageView image3 = (ImageView)findViewById(R.id.image3);

ImageView image4 = (ImageView)findViewById(R.id.image4);

Bitmap bmOriginal = BitmapFactory.decodeResource(getResources(), R.drawable.ic\_launcher);

image1.setImageBitmap(bmOriginal);

int width = bmOriginal.getWidth();

int height = bmOriginal.getHeight();

Bitmap bmDublicated2 = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);

Bitmap bmDublicated3 = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);

Bitmap bmDublicated4 = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);

int[] srcPixels = new int[width \* height];

bmOriginal.getPixels(srcPixels, 0, width, 0, 0, width, height);

int[] destPixels = new int[width \* height];

swapGB(srcPixels, destPixels);

bmDublicated2.setPixels(destPixels, 0, width, 0, 0, width, height);

image2.setImageBitmap(bmDublicated2);

swapRB(srcPixels, destPixels);

bmDublicated3.setPixels(destPixels, 0, width, 0, 0, width, height);

image3.setImageBitmap(bmDublicated3);

swapRG(srcPixels, destPixels);

bmDublicated4.setPixels(destPixels, 0, width, 0, 0, width, height);

image4.setImageBitmap(bmDublicated4);

}

void swapGB(int[] src, int[] dest) {

for (int i = 0; i < src.length; i++) {

dest[i] = (src[i] & 0xffff0000) | ((src[i] & 0x000000ff) << 8)

| ((src[i] & 0x0000ff00) >> 8);

}

}

void swapRB(int[] src, int[] dest) {

for (int i = 0; i < src.length; i++) {

dest[i] = (src[i] & 0xff00ff00) | ((src[i] & 0x000000ff) << 16)

| ((src[i] & 0x00ff0000) >> 16);

}

}

void swapRG(int[] src, int[] dest) {

for (int i = 0; i < src.length; i++) {

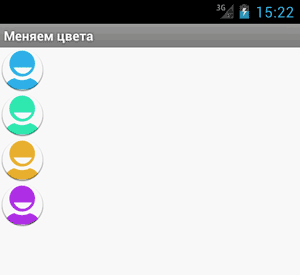
dest[i] = (src[i] & 0xff0000ff) | ((src[i] & 0x0000ff00) << 8)

| ((src[i] & 0x00ff0000) >> 8);

}

}

На скриншоте представлен оригинальный значок и три варианта замены цветов.



Ещё один пример, где также в цикле меняем цвет каждого пикселя Green->Blue, Red->Green, Blue->Red (добавьте на экран два ImageView):

public class BitmapProcessingActivity extends Activity {

ImageView imageView\_Source, imageView\_Dest;

Bitmap bitmap\_Source, bitmap\_Dest;

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

imageView\_Source = (ImageView)findViewById(R.id.source);

imageView\_Dest = (ImageView)findViewById(R.id.dest);

bitmap\_Source = BitmapFactory.decodeResource(getResources(), R.drawable.ic\_launcher);

imageView\_Dest.setImageBitmap(processingBitmap(bitmap\_Source));

}

private Bitmap processingBitmap(Bitmap src){

Bitmap dest = Bitmap.createBitmap(

src.getWidth(), src.getHeight(), src.getConfig());

for(int x = 0; x < src.getWidth(); x++){

for(int y = 0; y < src.getHeight(); y++){

// получим каждый пиксель

int pixelColor = src.getPixel(x, y);

// получим информацию о прозрачности

int pixelAlpha = Color.alpha(pixelColor);

// получим цвет каждого пикселя

int pixelRed = Color.red(pixelColor);

int pixelGreen = Color.green(pixelColor);

int pixelBlue = Color.blue(pixelColor);

// перемешаем цвета

int newPixel= Color.argb(

pixelAlpha, pixelBlue, pixelRed, pixelGreen);

// полученный результат вернём в Bitmap

dest.setPixel(x, y, newPixel);

}

}

return dest;

}

}

Конвертируем Bitmap в байтовый массив и обратно

public byte[] getByteArrayfromBitmap(Bitmap bitmap) {

ByteArrayOutputStream bos = new ByteArrayOutputStream();

bitmap.compress(CompressFormat.PNG, 0, bos);

return bos.toByteArray();

}

public Bitmap getBitmapfromByteArray(byte[] bitmap) {

return BitmapFactory.decodeByteArray(bitmap , 0, bitmap.length);

}

Как раскодировать Bitmap из Base64

Если изображение передаётся в текстовом виде через Base64-строку, то воспользуйтесь методом, позволяющим получить картинку из этой строки:

public static Bitmap convertBase64StringToBitmap(String source) {

byte[] rawBitmap = Base64.decode(source.getBytes(), Base64.DEFAULT);

Bitmap bitmap = BitmapFactory.decodeByteArray(rawBitmap, 0, rawBitmap.length);

return bitmap;

}

Вычисляем средние значения цветов

int[] averageARGB(Bitmap pic) {

int A, R, G, B;

A = R = G = B = 0;

int pixelColor;

int width = pic.getWidth();

int height = pic.getHeight();

int size = width \* height;

for (int x = 0; x < width; ++x) {

for (int y = 0; y < height; ++y) {

pixelColor = pic.getPixel(x, y);

A += Color.alpha(pixelColor);

R += Color.red(pixelColor);

G += Color.green(pixelColor);

B += Color.blue(pixelColor);

}

}

A /= size;

R /= size;

G /= size;

B /= size;

int[] average = {A, R, G, B};

return average;

}

Дополнительное чтение

[Displaying Bitmaps Efficiently](http://developer.android.com/training/displaying-bitmaps/index.html)