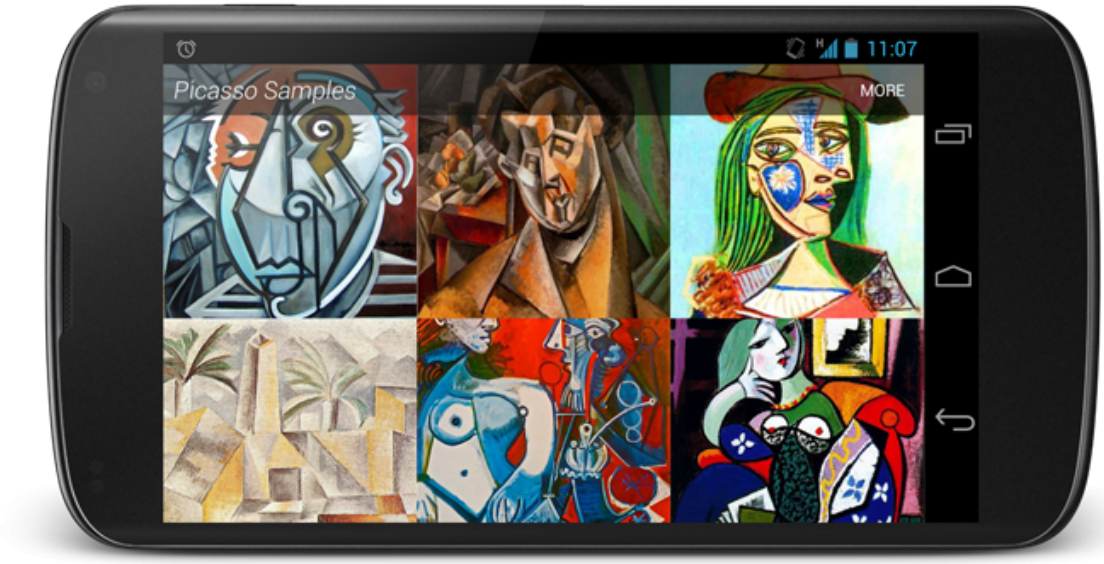
Android – Picasso



Introducción

Seguimos con una de esas librerías que nos solucionan de manera espectacular la utilización de imágenes en nuestros proyectos, Picasso. Existen otras como Glide (recomendada por Google) o Fresco (Facebook). Las diferencias entre Picasso y Glide son pocas, en cuanto a Fresco se refiere, es una aproximación diferente al tratamiento de imágenes en android. En este post (nearsoft) podeis ver una breve comparación entre la tres y en este otro post (inthecheesefactory blog) encontraras una comparación entre Glide y Picasso. Bien, vamos con Picasso:

- Configuración y utilización
- Picasso con assets o drawables
- Resize ,fit y rotation
- Scaling
- Placeholder y error
- Picasso Transformation Library
- Cache Indicators y Logging
- Observaciones

Configuración y utilización

Dentro del archivo app/build.gradle, debemos añadir la dependencia de Picasso.

```
dependencies{
  compile 'com.squareup.picasso:picasso:2.5.2'
}
```

Una vez añadida podemos empezar a utilizarla, como podéis ver es muy simple:

```
ImageView ourImageView = (ImageView) findViewById(R.id.imageView);
String remoteUrl = "http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg";

Picasso
    .with(this)
    .load(remoteUrl)
    .into(ourImageView);
```

Picasso con assets, drawables o ficheros guardados

Realmente fácil:

```
// Loading drawable
Picasso
    .with(this)
    .load(R.drawable.image)
    .into(imageView1);

// Loading asset
Picasso
    .with(this)
    .load("file:///android_asset/image.png")
    .into(imageView2);

// Loading file from storage
```

```
File file = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), "Android.png");
Picasso
    .with(this)
    .load(file)
    .into(imageView3);
```

Resize, fit y rotation

Podemos redimensionar la imagen.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg")
    .resize(100, 100)
    .into(imageView);
```

Pero en caso de que la imagen sea más pequeña tenemos la opción de evitar este reescalado. Redimensionar una imagen haciéndola más grande nos supone un uso de recursos que muchas veces nos va a dar un resultado muy pobre. En este caso podemos utilizar **scaleDown(true)**, de esta manera la imagen se redimensionará si el resultado final implica unas dimensiones inferiores a las originales.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg")
    .resize(100, 100)
    .scaleDown()
    .into(imageView);
```

O bien hacer que se redimensione automáticamente al tamaño del imageView. Esto puede hacer que la carga de la imagen tarde un poco más, puesto que primero se debe esperar a poder obtener el tamaño del imageView.

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .fit()
    .into(imageView);
```

Podemos rotar la imagen

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .rotate(180f)
    .into(imageView);
```

E incluso indicar que punto queremos utilizar para pivotar la rotación

```
// rotate(float degrees, float pivotX, float pivotY)
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .rotate(45f, 200f, 100f)
    .into(imageViewComplexRotate);
```

Scaling

El reescalado de imágenes puede afectar el aspect ratio y hacer que esta se vea deforme. Para solucionar esto podemos utilizar centerCrop() o centerInside().

CenterCrop
Se escala la imagen haciendo que coincida con los límites del ImageView y entonces se elimina la parte restante de la imagen. El ImageView contendrá la imagen pero seguramente se perderán partes de esta.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
```

```
")
    .resize(100, 100) // resizes the image to these dimensions
(in pixel)
    .centerCrop()
    .into(imageViewResizeCenterCrop);
```

CenterInside
Se escala la imagen teniendo en cuenta que las dos dimensiones de la imagen son iguales o inferiores al tamaño del imageView. La imagen se mostrara completa pero puede que no ocupe todo el imageView.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .resize(100, 100)
    .centerInside()
    .into(imageViewResizeCenterInside);
```

Placeholder y error

Podemos utilizar una imagen temporal hasta que nuestra imagen se haya cargado y otra en caso de que ocurra un error. De esta manera, el usuario tendrá la sensación que durante un tiempo de espera o incluso un error la aplicación funciona perfectamente.

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .placeholder(R.drawable.placeholder_image)
    .error(R.drawable.error_image)
    .into(imageView);
```

Picasso Transformation Library

Existe una librería que nos permite hacer transformaciones a

nivel avanzado y de manera muy fácil picasso-transformations. Os recomiendo que paséis por su cuenta de Github puesto que aquí solo mostraremos dos ejemplos.

Primeramente en el archivo app/build.gradle, debemos añadir la dependencia de picasso-transformations.

```
dependencies{
    compile 'jp.wasabeef:picasso-transformations:2.1.0'
}
```

Una vez añadida podemos empezar a tulizarla, por ejemplo para aplicar un crop circular:

```
    Picasso
    .with(this)

.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .transform(new CropCircleTransformation())
    .into(imageView);
```

También podemos encadenar transformaciones, por ejemplo haciendo un crop como en el ejemplo anterior y aplicando un filtro de color:

```
int color = Color.parseColor("#3300ff80");
Picasso
    .with(this)
    .load("http://www.guitarthai.com/picpost/gtpicpost/Q367224
.jpg")
    .transform(new ColorFilterTransformation(color))
    .transform(new CropCircleTransformation())
    .into(imageView);
```

Cache Indicators y Logging

Cache Indicators
Picasso utiliza dos tipos de chache, memoria y almacenamiento. En algunos casos, para comprobar y hacer tests de rendimiento

de nuestra aplicación, nos interesará saber de donde se ha obtenido la imagen. Para hacerlo debemos añadir la opción **.setIndicatorsEnabled(true)**

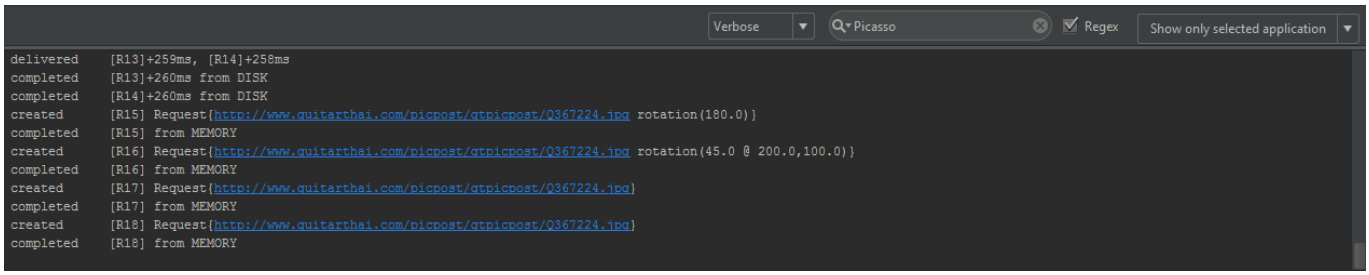
```
Picasso
    .with(this)
    .setIndicatorsEnabled(true);
Picasso
    .with(this)
    .setIndicatorsEnabled(true);
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .into(imageView);
```

Una vez cargadas la imágenes estas tendrán un indicador de color en la parte superior izquierda. El esquema de colores corresponde al origen del cual la imagen es:

- Verde (memoria, mejor rendimiento)
- Azul (memoria interna del dispositivo, rendimiento medio)
- Rojo (red, el peor rendimiento)

Logging
Aveces necesitamos más información que una simple indicación del origen de la imagen. Utilizando la opción **.setLoggingEnabled(true);** obtendremos en el log una información más acurda del proceso creado por Picasso.

```
Picasso
    .with(this)
    .setLoggingEnabled(true);
Picasso
    .with(this)
    .setLoggingEnabled(true);
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .into(imageView);
```



Observaciones

Bien, hoy hemos tratado una librería la cual conviene tener en la caja de herramientas. Nos ofrece muchas opciones y nos permite realizar operaciones de una manera muy simple. Como contrapartida podemos decir que el rendimiento siempre se verá afectado. Finalmente, dejo el enlace a la web de Picasso y un simple ejemplo en Github.

- Picasso
- PicassoTest