

●グラフィック描画と View クラス

android ではどうやってグラフィックを描画するのでしょうか。
専用のクラスを定義し、それを画面にはめ込んで表示します。専用クラスの中には、描画のためのメソッドを用意して、そこで描画の処理を実行する。そういうスタイルです。

グラフィック表示用に用意するクラスは、一般に「View」クラスのサブクラスとして作成します。

View というのは、画面に表示される部品類（ビュー）の基本となるクラスです。このクラスを継承したクラスを用意し、それを Activity クラスに組み込みます。この組み込み処理は、いつもの「setContentView()」メソッドを使って行います。

View のサブクラスは、だいたい以下のような形で定義します。

```
class クラス extends View {  
  
    public コンストラクタ(Context context) {  
        super(context);  
        .....初期化処理.....  
    }  
  
    public void onDraw(Canvas c) {  
        .....描画の処理.....  
    }  
  
}
```

スーパークラスである View には、Context を引数に持つコンストラクタしか用意されておらず、引数なしのデフォルトコンストラクタはありません。

したがって、必ず Context を引数に持つコンストラクタを定義する必要があります。これは、特に処理がなければ super(context); とするだけでかまいません。

実際の描画処理を用意するのが、「onDraw()」メソッドです。これは「Canvas」というクラスが引数に渡されます。

グラフィック描画の雛形を作る

では、実際に簡単なサンプルを作ってみましょう。とりあえずは、何も表示しない、プログラムの骨格部分を作成してみます。

ここでは DrawView というクラスを定義し、これを表示させるようにしておきましょう。

```
package jp.allabout.android;

import android.app.Activity;
import android.content.Context;
import android.graphics.*;
import android.os.Bundle;
import android.view.View;

public class GraphicsTestActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        DrawView view = new DrawView(this);
        setContentView(view);
    }
}

class DrawView extends View {
    //コンストラクタ
    public DrawView(Context context) {
        super(context);
    }
}
```

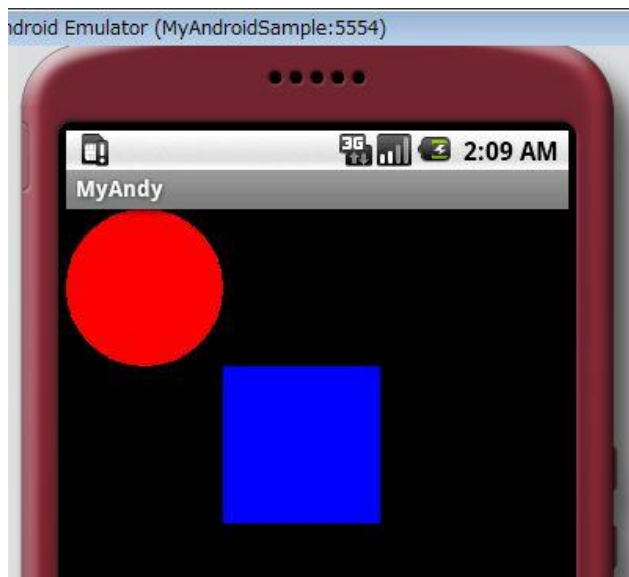
```
public void onDraw(Canvas c){  
    // ここに描画処理を追加  
}  
  
} //class_end
```

これを実行しても、画面は真っ暗でまだ何も表示はされません。とりあえず、これで「入れ物」はできました。これに追加しながら描画を行っていきましょう。

色の設定と図形の描画

では、実際に簡単な図形を描画させて見ることにします。DrawView の onDraw メソッドを以下のように修正してみましょう。

```
public void onDraw(Canvas c){  
  
    c.drawColor(Color.BLACK);  
  
    Paint fill_paint = new Paint();  
    fill_paint.setStyle(Paint.Style.FILL);  
    fill_paint.setColor(Color.BLUE);  
    c.drawRect(100f, 100f, 200f, 200f, fill_paint);  
  
    fill_paint.setColor(Color.RED);  
    c.drawOval(new RectF(0f, 0f, 100f, 100f), fill_paint);  
  
}
```



ここでは、赤と青の円・四角形をそれぞれ描画してみました。とりあえずは、図形描画の基本はこれでわかることでしょう。

ここでは、大きく分けて2つのオブジェクトが登場します。1つは、onDraw で引数に渡される「Canvas」、そしてもう1つは「Paint」です。

Canvas は、そのビューへの描画を行うための機能を提供するクラスです。ここに用意されている描画関係のメソッドを呼び出すことで図形を描画することができます。これは Graphics クラスとほぼ同じですね。

では、Paint は何か？ これは、描画(特に、塗りつぶし方)に関する各種の設定情報を管理するクラスだと考えてください。Swing などでも Graphics2D では同様のクラスが登場するので、ぴんと来た人もいるかも知れませんね。

描画を行う場合、まず、塗り方(図形の塗り方の方式や塗りつぶす色に関する設定など)を Paint インスタンスとして用意します。そして、Canvas にある描画メソッドを呼び出して描画を行い、このとき用意した Paint インスタンスを渡してやります。これで、Paint で指定したやり方で図形が描かれる、というわけです。

では、実行している処理を見ていきましょう。まず最初に、こういうことをやってますね。

```
c. drawColor (Color. BLACK) ;
```

この `drawColor` は、この `View` の背景色を指定するものです。これで最初に背景を設定します。ここでは「`Color.BLACK`」という値を使っていますね。色の値というのは、android では `int` 値で指定されるのですが、これは `Color` クラスに用意されているクラスフィールドや、色の値を返すメソッドなどを使って指定します。ここでの `BLACK` のように、主な色はすべて `Color` クラスに用意されているので、まずはそれを利用していくとよいでしょう。

```
Paint fill_paint = new Paint();
```

次に、描画に使う `Paint` インスタンスを作成します。これは、そのまま `new` するだけです。

```
fill_paint.setStyle(Paint.Style.FILL);
```

描画スタイルを設定します。ここでは「`Paint.Style.FILL`」を指定し、図形の内部を塗りつぶす方式を選びます。他、輪郭線だけを描く `STROKE`、塗りつぶしと輪郭線の両方を描く `FILL_AND_STROKE` などがあります。

```
fill_paint.setColor(Color.BLUE);
```

`Paint` に色の設定を行います。`setColor` で、色の値を指定するだけです。

```
c.drawRect(100f, 100f, 200f, 200f, fill_paint);
```

四角形を描画します。`drawRect` は、引数の値をもとに四角形を描きます。引数には、左・上・右・下の位置と使用する `Paint` オブジェクトを指定します。

```
fill_paint.setColor (Color. RED) ;  
c.drawOval (new RectF (0f, 0f, 100f, 100f), fill_paint) ;
```

再び Paint の setColor を呼び出して私用する色を変更し、「drawOval」で円を描きます。これは、描く領域を示す値 (RectF インスタンス) と使用する Paint インスタンスを引数に渡して呼び出します。

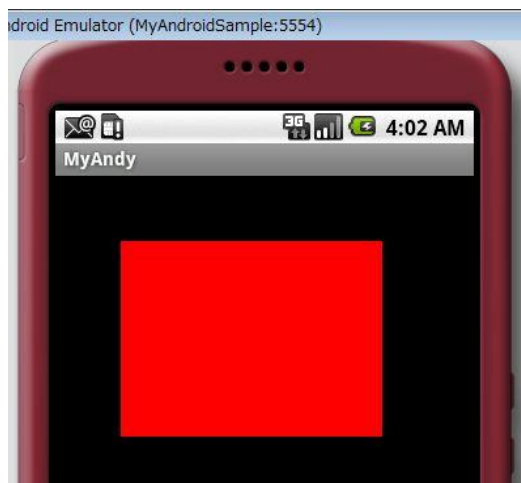
領域を指定する RectF というのは、左・上・右・下の位置を引数で渡してインスタンスを作成します。

さまざまな図形の描画

では、さまざまな図形を描く描画メソッドをざっとここで整理していきましょう。

・四角形を描く

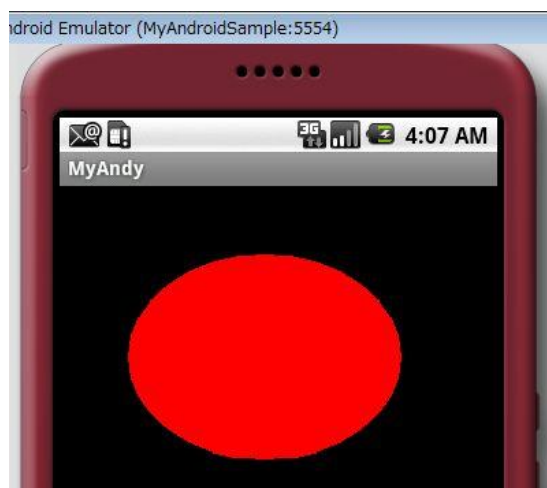
```
drawRect ( 左位置, 上位置, 右位置, 下位置, [Paint] )  
drawRect ( [Rect], [Paint] )  
drawRect ( [RectF], [Paint] )
```



drawRect の描画例。

・円を描く

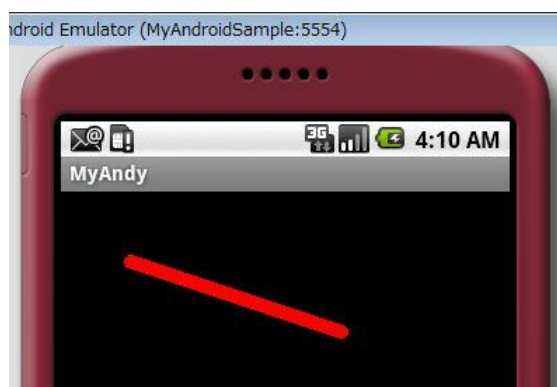
```
drawOval( 左位置, 上位置, 右位置, 左位置, [Paint] )  
drawCircle( 中心横, 中心縦, 半径, [Paint] )
```



drawOval の描画例。

・直線を描く

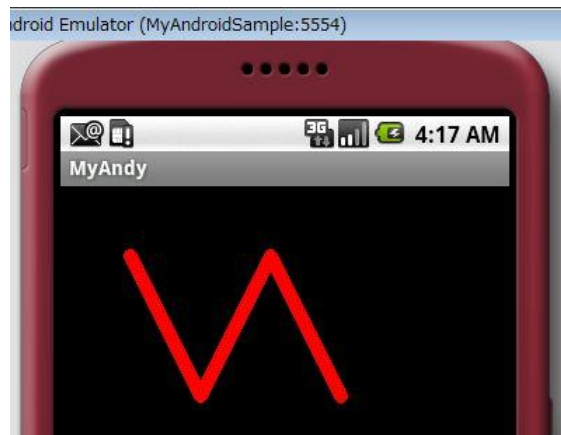
```
drawLine( 横 1, 縦 1, 横 2, 縦 2, [Paint] )
```



drawLine の描画例。

・多角形を描く

```
drawLines( float 配列, [Paint] )
```



drawLines の描画例。

・テキストを描く

```
drawText( String 値, 横位置, 縦位置, [Paint] )
```



drawText の描画例。

次にサンプルコードを示します。

```
// 描画処理をここに記述する
@Override
protected void onDraw(Canvas canvas) {

    Paint paint = new Paint();
    paint.setColor(Color.argb(255, 255, 255, 255));

    // 点を描く
    canvas.drawPoint(40, 50, paint);

    // 複数の点をまとめて描く。2点を配列で指定も可。
    float[] pts = { 20, 30, 30, 40 };
    canvas.drawPoints(pts, paint);

    canvas.drawLine(0, 0, 10, 20, paint);

    // 配列で2点のxy座標を記述することも可
    float[] pts2 = { 50, 0, 50, 30 };
    canvas.drawLines(pts2, paint);

    // 矩形
    Rect rect = new Rect(50, 50, 80, 80);
    canvas.drawRect(rect, paint);

    RectF rectF = new RectF(40.5f, 20.5f, 60.5f, 40.5f);
    canvas.drawRect(rectF, paint);

    paint.setStyle(Style.STROKE); // 枠線
    canvas.drawRect(10, 50, 30, 80, paint);

    // 円
    paint.setStyle(Style.FILL); // 塗りつぶし
    canvas.drawCircle(100.5f, 100.5f, 30.0f, paint);

    //アンチエイリアス（境界線のドットが、バックの色と融合している）
    //アンチエイリアスしないもの（境界線のドットがはっきりとしている）
```

```
        paint.setAntiAlias(true);

        RectF ovalF = new RectF(200f, 200f, 100f, 50f);
        canvas.drawOval(ovalF, paint);

        //文字列
        paint.setTextSize(20);
        canvas.drawText("こんにちは。", 100, 350, paint);
    }
} //onDraw()
```

イメージリソースの表示

更に複雑なグラフィックを表示させたい場合には、あらかじめビットマップイメージファイルをリソースに用意し、これをロードして描画させる、というやり方をとります。これも実際にやってみましょう。

まず、イメージファイルを作成します。android では、JPEG、PNG、GIF といったイメージを利用することができます。ここでは「image.jpg」という JPEG イメージファイルを作成することにしましょう。

イメージは、「res」内の「drawable」というフォルダの中にコピーしておきます。ここには、既に「icon.png」というアイコンのイメージが用意されていますね。描画に関するイメージは、ここにまとめておくのが基本です。

では、このイメージを実際に画面に描画してみましょう。

```
public class MyAndy extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

        DrawView view = new DrawView(getApplicationContext());
        setContentView(view);
    }
}

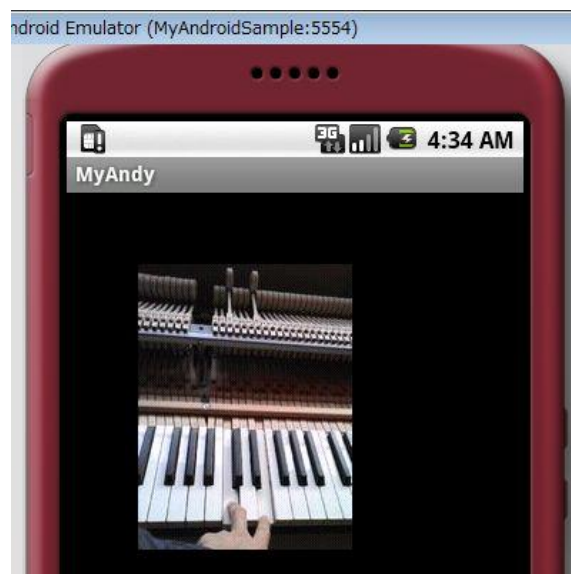
class DrawView extends View {
    private Bitmap img;

    public DrawView(Context context) {
        super(context);
        Resources res = this.getContext().getResources();
        img = BitmapFactory.decodeResource(res, R.drawable.image);
    }

    public void onDraw(Canvas c) {
        c.drawColor(Color.BLACK);

        Paint fill_paint = new Paint();
        c.drawBitmap(img, 50, 50, fill_paint);
    }
}

```



イメージリソースの描画例。

ここでは、「Bitmap」というフィールドを用意していますね。これが、ビットマップイメージのクラスになります。

```
Resources res = this.getContext().getResources();
```

まず、リソースを扱うためのクラス「Resources」を用意します。これは Activity クラスの `getContext` で得られる Context クラスに用意されている「`getResources`」を使います。

```
img = BitmapFactory.decodeResource(res, R.drawable.image);
```

続いて、`BitmapFactory` というクラスにある「`decodeResource`」メソッドを呼び出します。これは、指定されたリソースを読み込み、`Bitmap` オブジェクトに変換して返すものです。引数には、`Resources` インスタンスと、読み込むイメージの番号を指定します。番号 `D` は、例によって `R` クラスの `drawable` から取り出します。ここに、`image.jpg` の番号が自動生成されているはずです（`drawable` には、「`drawable`」フォルダに入れてあるファイル名のフィールドが自動的に用意されます）。

これでリソースからイメージデータを取り出し、`Bitmap` として取得するところまでできました。後は、`onDraw` 内でこれを描画するだけです。

```
c.drawBitmap(img, 50, 50, fill_paint);
```

「`drawBitmap`」が、そのためのメソッドです。これは引数に `Bitmap` インスタンス、描画する横・縦位置、`Paint` インスタンスをそれぞれ渡します。横・縦位置の代わりに、`RectF` を渡すことで、指定した領域内にイメージを描画させることなども可能です。

画面操作で画像を動かす Android アプリを作成

●MovingActivity.java

```
import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;

public class MovingActivity extends Activity {
    private SampleView mView;
    private Bitmap mBitmap;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mBitmap = BitmapFactory.decodeResource(getResources(),
                                                    R.drawable.icon);

        mView = new SampleView(this);
        setContentView(mView);
    }

    private class SampleView extends View {
        private Paint mPaint;
        //画像表示の左上角の x y 座標を示す変数
        private float imageX = 0f;
        private float imageY = 0f;
```

```
public SampleView(Context context) {
    super(context);
    mPaint = new Paint();
}

//描画処理を記述
@Override
protected void onDraw(Canvas canvas) {
    //canvas のカラー
    canvas.drawColor(Color.WHITE);
    //アンチエイリアス
    mPaint.setAntiAlias(true);
    //カラー設定（デフォルトは BLACK）
    mPaint.setColor(Color.BLUE);

    //テキスト表示
    canvas.drawText(getString(R.string.hello), 0, 20, mPaint);

    //画像表示
    canvas.drawBitmap(mBitmap,
        imageX - mBitmap.getWidth() / 2,
        imageY - mBitmap.getHeight() / 2,
        mPaint);
}

//タッチイベントの動作を記述
@Override
public boolean onTouchEvent(MotionEvent event) {

    //触る
    if(event.getAction() == MotionEvent.ACTION_DOWN) {
        imageX = event.getX();
        imageY = event.getY();
    }
    //触ったままスライド
    else if(event.getAction() == MotionEvent.ACTION_MOVE) {
```

```

        imageX = event.getX();
        imageY = event.getY();
    }

    // 離す
    else if(event.getAction() == MotionEvent.ACTION_UP) {
        imageX = event.getX();
        imageY = event.getY();
    }

    // 再描画の指示
    invalidate();

    return true;
}
}
}

```

コードの解説)

今回は View クラスを使用して画面表示をしています。
 画像表示には android.graphics.Bitmap を使用しています。

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mBitmap = BitmapFactory.decodeResource(getResources(),
                                                R.drawable.star);    mView =
    new SampleView(this);
    setContentView(mView);
}

```

onCreate 内で

android.graphics.BitmapFactory.decodeResource(Resources res, int id)
 を使用し、Bitmap を作成しています。ID は star.png の ID を指定しています。

```
mView = new SampleView(this);  
setContentView(mView);
```

setContentView()にはViewクラスを指定しています。

SampleView クラスの描画処理

```
@Override  
protected void onDraw(Canvas canvas) {  
  
    //canvas のカラー  
    canvas.drawColor(Color.WHITE);  
    //アンチエイリアス  
    mPaint.setAntiAlias(true);  
    //カラー設定 (デフォルトは BLACK)  
    mPaint.setColor(Color.BLUE);  
  
    //テキスト表示  
    canvas.drawText(getString(R.string.hello), 0, 20, mPaint);  
  
    //画像表示  
    canvas.drawBitmap(mBitmap,  
        imageX - mBitmap.getWidth() / 2,  
        imageY - mBitmap.getHeight() / 2,  
        mPaint);  
}
```

onDraw(Canvas canvas)

で画面の表示をしています。

```
canvas.drawColor(Color.WHITE);
```

で canvas の色（背景色）を指定しています。

```
mPaint.setAntiAlias(true);
```

でアンチエイリアス設定をしています。

デフォルトは false であるため、アンチエイリアスがかかりません。

```
mPaint.setColor(Color.BLUE);
```

カラーを設定しています。

以降に記述しているテキストなどの色に反映されます。

```
canvas.drawText(getString(R.string.hello), 0, 20, mPaint);
```

```
drawText (String text, float x, float y, Paint paint)
```

を使用してテキスト表示の設定をしています。

```
getString (int resId)
```

で ID を指定し、string 型のデータを取得しています。

```
        canvas.drawBitmap(mBitmap,
                           imageX - mBitmap.getWidth() / 2,
                           imageY - mBitmap.getHeight() / 2,
                           mPaint);
```

```
drawBitmap (Bitmap bitmap, float left, float top, Paint paint)
```

を使用して画像表示の設定をしています。

```
left = 0f
```

```
top = 0f
```

を指定すると、画像の (0, 0) 位置が画面の (0, 0) に表示されます。

今回はそれぞれ

```
- mBitmap.getWidth() / 2
```

```
- mBitmap.getHeight() / 2
```

を設定しているため、

```
imageX = 0f
```

```
imageY = 0f
```

の場合は画像の (0, 0) 位置が画面の

(- mBitmap.getWidth() / 2, - mBitmap.getHeight() / 2) に表示されます。

SampleView クラスのタッチイベント処理

```
@Override
```

```
public boolean onTouchEvent(MotionEvent event) {
```

```
    //触る
```

```
    if(event.getAction() == MotionEvent.ACTION_DOWN) {
```

```
        imageX = event.getX();
```

```
        imageY = event.getY();
```

```
    }
```

```
    //触ったままスライド
```

```
    else if(event.getAction() == MotionEvent.ACTION_MOVE) {
```

```
        imageX = event.getX();
```

```
        imageY = event.getY();
    }
    //離す
    else if(event.getAction() == MotionEvent.ACTION_UP) {
        imageX = event.getX();
        imageY = event.getY();
    }

    // 再描画の指示
    invalidate();

    return true;
}
```

今回は `onTouchEvent(MotionEvent event)` を使用して、画面がタッチされたときの処理を行っています。

```
event.getX();
```

```
event.getY();
```

でタッチされた画面の座標を取得し、画像の表示位置に反映されるように記述しています。

```
invalidate();
```

で `onDraw(Canvas canvas)` が呼ばれ、画面が再描画されます。
