第11章

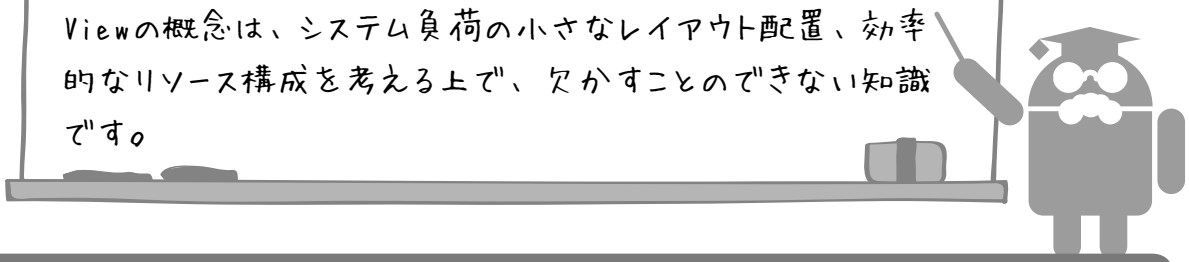
UIのカスタマイズ

著：日高正博

11-1 Viewの役割

著：日高正博

AndroidのViewシステムについて解説します。Androidでは、すべての画面がViewという要素によって構成されています。Viewの概念は、システム負荷の小さなレイアウト配置、効率的なリソース構成を考える上で、欠かすことのできない知識です。



この節で学ぶこと

- AndroidのViewやUIコンポーネントの知識
- レイアウトの仕組み
- Viewをカスタマイズして機能を追加する手法

この節で出てくるキーワード一覧

フォーカス
リスナー
プロパティ
UIコンポーネント
タッチイベント
View
ViewGroup
LayoutInflater
Path
Canvas

11-1-1 フォーカスの設定

フォーカスとは、画面上でユーザーが操作できるようになっている部分のこと。テキストボックスなら、カーソルが点滅して、「ここに文字を入力できます」ということを示しています。

Androidはユーザーの入力に応じて「フォーカス」を自動的に移動します。特定のViewへフォーカスを指定したい場合は「requestFocus」メソッドを使います。

Viewの表示設定は「setVisibility」メソッドを使うことでViewの表示／非表示状態を変更できます。それぞれViewを継承した各UIコンポーネントでも利用できるメソッドです。

プロパティは、ソースコードまたはXMLから可能です。たとえば以下の2つのコードのとおり、「TextView」では「setText」メソッドと「android:text」属性があります。

プロパティとは、オブジェクトの保持しているデータのこと。Viewであれば高さや幅が必要など、オブジェクトの性質を表すものが一般的。

buttonのプロパティを設定する（Java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button btn  = (Button) findViewById(R.id.button);
    btn.setText("Hello!");
}
```

ソースコードから文字列を設定するには「setText」メソッドが利用できます。XMLからは「android:text」属性で文字列を設定できます。

buttonのプロパティを設定する（XML）

```
<RelativeLayout ～省略～
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:text="Hello!" />
</RelativeLayout>
```

このようにJavaのコードやXMLを使って、Viewを組み合わせることで画面が形作られています。

ソースコードを利用したほうが表現力の幅は広がり、できることも多いですが、レイアウトがいろんなところから参照されることで複雑度が格段にあがります。

基本的にはXMLで構築し、制御が必要な場合にのみソースコードから参照するようにしましょう。

最後のリスナーの設定は、Viewに変化があった場合に通知する仕組みです。

Viewには、フォーカスに変化があったときに、呼び出されるリスナーを登録する「setOnFocusChangeListener」メソッド、Viewがクリックされたときに呼び出されるリスナーを登録する「setOnClickListener」メソッド、タッチイベントが発生したときに呼び出されるリスナーを登録する「setOnTouchListener」メソッドなどが用意されています。

11-1-2 UIコンポーネントのXML属性と関連するメソッド

ここではViewのXML属性と関連するメソッドを解説します。**表1**で挙げたViewの属性値は、UIコンポーネント（「TextView」や「ImageViewなど」）でも共通です。

XML 要素	関連メソッド	概要
android:accessibilityLiveRegion	setAccessibilityLiveRegion(int)	アクセシビリティサービスへの通知種類
android:alpha	setAlpha(float)	View の透明度
android:background	setBackgroundResource(int)	背景に設定する drawable を指定
android:clickable	setClickable(boolean)	クリックの有効判定
android:contentDescription	setContentDescription(CharSequence)	コンテンツの概要
android:drawingCacheQuality	setDrawingCacheQuality(int)	キャッシュの品質
android:duplicateParentState	なし	子ビューの状態を共有
android:fadeScrollbars	setScrollbarFadingEnabled(boolean)	フェードの有効判定
android:fadingEdgeLength	getVerticalFadingEdgeLength()	フェードアウトする領域の幅
android:filterTouchesWhenObscured	setFilterTouchesWhenObscured(boolean)	タッチフィルタリング
android:fitsSystemWindows	setFitsSystemWindows(boolean)	レイアウトでシステムウィンドウを考慮する
android:focusable	setFocusable(boolean)	フォーカスの有効判定
android:focusableInTouchMode	setFocusableInTouchMode(boolean)	タッチモード時のフォーカスの有効判定
android:hapticFeedbackEnabled	setHapticFeedbackEnabled(boolean)	触覚フィードバック
android:id	setId(int)	findViewById で取得できる識別子
android:importantForAccessibility	setImportantForAccessibility(int)	アクセシビリティで特別に必要な場合
android:isScrollContainer	setScrollContainer(boolean)	IME が表示される場合のスクロール設定
android:keepScreenOn	setKeepScreenOn(boolean)	表示中はスクリーンを ON
android:layerType	setLayerType(int,Paint)	ハードウェアレイヤが指定可能
android:layoutDirection	setLayoutDirection(int)	アラビア語圏など RTL 設定
android:longClickable	setLongClickable(boolean)	長押し有効判定
android:minHeight	setMinimumHeight(int)	高さの最小値
android:minWidth	setMinimumWidth(int)	横幅の最小値
android:nextFocusDown	setNextFocusDownId(int)	FOCUS_DOWN 移動時の遷移先 View
android:nextFocusForward	setNextFocusForwardId(int)	FOCUS_FORWARD 移動時の遷移先 View
android:nextFocusLeft	setNextFocusLeftId(int)	FOCUS_LEFT 移動時の遷移先 View
android:nextFocusRight	setNextFocusRightId(int)	FOCUS_RIGHT 移動時の遷移先 View
android:nextFocusUp	setNextFocusUpId(int)	FOCUS_UP 移動時の遷移先 View
android:onClick	なし	クリック時のメソッド呼び出し
android:padding	setPaddingRelative(int,int,int,int)	パディングの設定
android:paddingBottom	setPaddingRelative(int,int,int,int)	パディング（下）の設定
android:paddingEnd	setPaddingRelative(int,int,int,int)	パディング（終端）の設定
android:paddingLeft	setPadding(int,int,int,int)	パディング（左）の設定
android:paddingRight	setPadding(int,int,int,int)	パディング（右）の設定
android:paddingStart	setPaddingRelative(int,int,int,int)	パディング（始端）の設定
android:paddingTop	setPaddingRelative(int,int,int,int)	パディング（上）の設定
android:requiresFadingEdge	setVerticalFadingEdgeEnabled(boolean)	フェード指定
android:rotation	setRotation(float)	View の回転角
android:rotationX	setRotationX(float)	View の回転角（X 軸）
android:rotationY	setRotationY(float)	View の回転角（Y 軸）
android:saveEnabled	setSaveEnabled(boolean)	状態保存の有効判定
android:scaleX	setScaleX(float)	拡大率（X 軸）
android:scaleY	setScaleY(float)	拡大率（Y 軸）
android:scrollX	なし	スクロールの水平オフセット位置
android:scrollY	なし	スクロールの垂直オフセット位置
android:scrollbarAlwaysDrawHorizontalTrack	なし	水平方向の常時表示
android:scrollbarAlwaysDrawVerticalTrack	なし	垂直方向の常時表示
android:scrollbarDefaultDelayBeforeFade	setScrollBarDefaultDelayBeforeFade(int)	消え始めるまでの時間
android:scrollbarFadeDuration	setScrollBarFadeDuration(int)	消えるまでの時間
android:scrollbarSize	setScrollBarSize(int)	スクロールバーのサイズ
android:scrollbarStyle	setScrollBarStyle(int)	スクロールバーのスタイル指定
android:scrollbarThumbHorizontal	なし	水平方向の drawable 指定（色を変更）
android:scrollbarThumbVertical	なし	垂直方向の drawable 指定（色を変更）
android:scrollbarTrackHorizontal	なし	水平方向の drawable 指定（トラックを変更）
android:scrollbarTrackVertical	なし	垂直方向の drawable 指定（トラックを変更）
android:scrollbars	なし	スクロールバーの表示設定
android:soundEffectsEnabled	setSoundEffectsEnabled(boolean)	効果音の設定
android:tag	なし	タグ名
android:textAlignment	setTextAlignment(int)	テキストの揃え
android:textDirection	setTextDirection(int)	テキストの表示方向
android:transformPivotX	setPivotX(float)	反転方向（X 軸）
android:transformPivotY	setPivotY(float)	反転方向（Y 軸）
android:translationX	setTranslationX(float)	移動（X 軸）
android:translationY	setTranslationY(float)	移動（Y 軸）
android:visibility	setVisibility(int)	表示の初期状態

ViewのXML属性と関連するメソッド

システムとして、共通のアクセシビリティ(ユーザー補助機能)、表示に関連したスクロール機能、レイアウト上必要となるパディング、操作に必要なフォーカスなどが設定できます。またViewを加工する回転、移動など表示に必要な設定項目を幅広く持っていることがわかります。

```
java.lang.Object
 └android.view.View
   └android.widget.TextView
```

たとえば「TextView」は、前述のクラス継承関係に基づいて、Viewを継承しています。さらに「TextView」独自のメソッドやXML属性を持っています。

TextView独自のメソッドやXML属性は以下のURLを参照してください(http://developer.android.com/reference/android/widget/TextView.html)。

UIコンポーネントの「Button」をみてみると、「TextView」を継承しているため、「TextView」に加えて「Button」独自のメソッドを持っています。

```
java.lang.Object
 └android.view.View
   └android.widget.TextView
     └android.widget.Button
```

このようにViewを継承し、機能を追加、設定してコンポーネントを構成しています。

「Button」のメソッドは以下のURLを参照してください(http://developer.android.com/reference/android/widget/Button.html)。

11-1-3 ViewGroupの役割

「FrameLayout」や「RelativeLayout」といったレイアウトは、たくさんのViewを持ち、開発者の意図した場所に配置する役割を担っています。そのための仕組みとして「ViewGroup」があります。

FrameLayout: http://developer.android.com/reference/android/widget/FrameLayout.html

RelativeLayout: http://developer.android.com/reference/android/widget/RelativeLayout.html

画面上に、たった1つのUIコンポーネントが存在しているという構成はあまりありません。テキスト、画像、ボタンを配置して、画面を構成することが一般的です。

「ViewGroup」は構成要素としてUIコンポーネントを持てる特殊なクラスです。構成要素として持っているViewのことを「子View(children)」、保持している側を「親(parent)」と呼びます。

「子View」「親」だけでなく、ツリー構造など関係の主従を意識した呼び方がされることがあります。しかし方言が多いので決まった言い方ではありません。

ここで、FrameLayoutのクラス継承関係をみてみましょう。

```
java.lang.Object
 └android.view.View
   └android.view.ViewGroup
     └android.widget.FrameLayout
```

「FrameLayout」が「ViewGroup」を継承していることがわかります。さらに「View」を継承しているため、自分自身も「View」であるといえます。そのため「Layout」そのものを「Layout」に内包することもでき、**図1**のようなツリー構造で書き表せます。

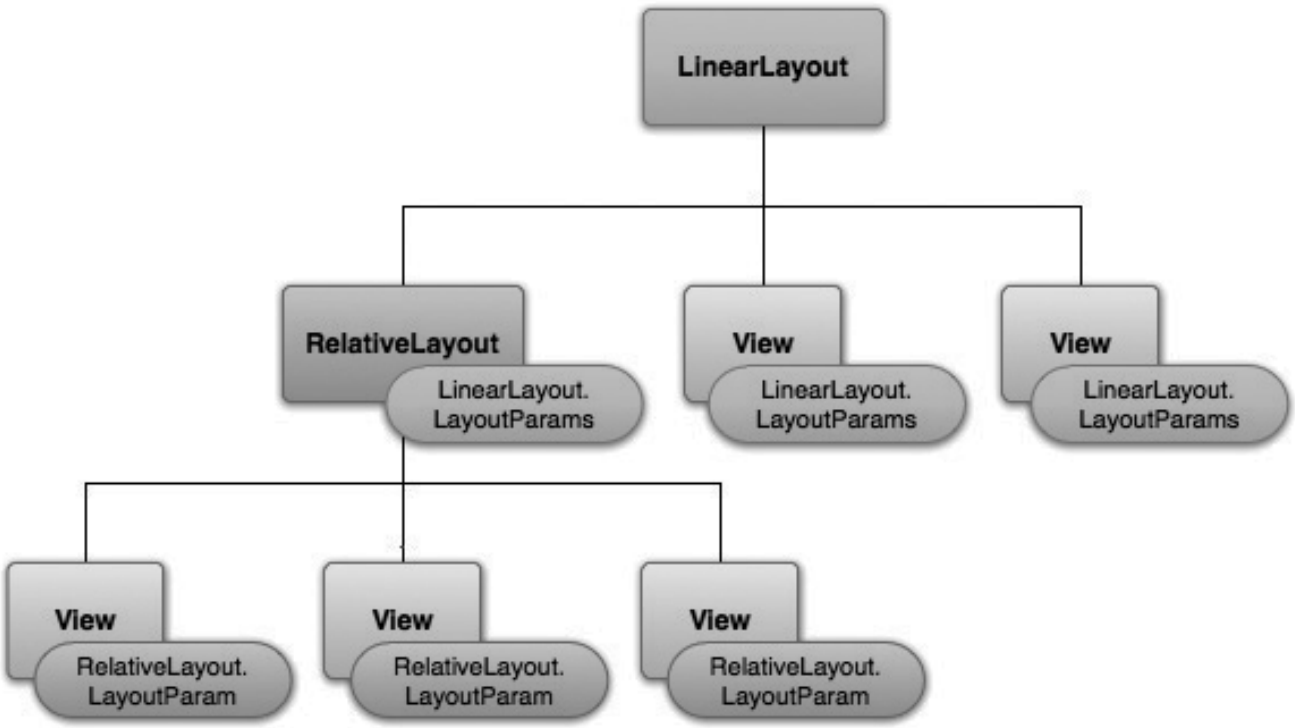


図1: レイアウトの構造（出典：http://developer.android.com/guide/topics/ui/declaring-layout.html）

このような構造をツリー構造と呼んだり「View」が「View」のなかに入ることから、入れ子構造と呼んだりします（**図2**）。

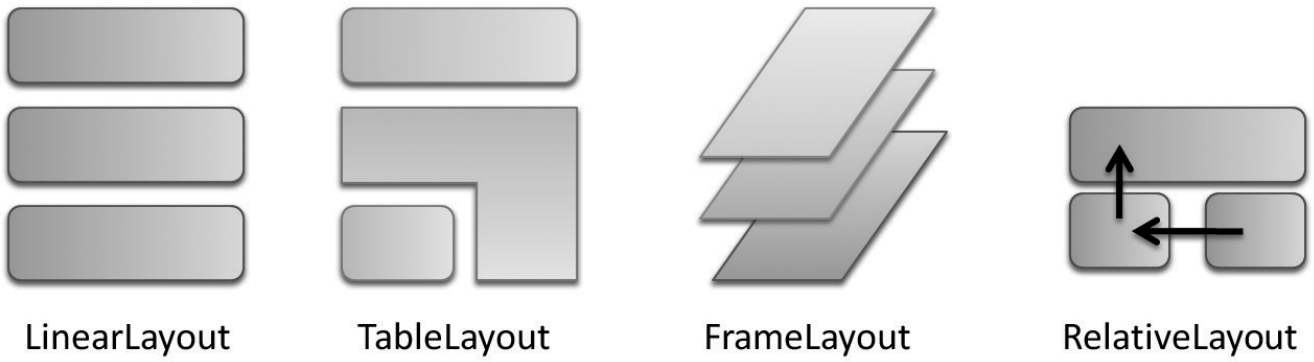


図2: レイアウトは「View」を複数保持できる

それぞれの「View」が、お互いの位置関係を把握、自分の表示位置を計算するという順序で描画されます。

①Viewの位置、内容を計測する
②Viewを実際に描画する

こうした描画処理に関して、最終的な責任は、Activityが持っています。「View」のツリー構造の「root（最も重要なベース）」として、表示するレイアウトや「View」を切り替える責務を持っています。端的にこの部分を表しているソースコードが次のサンプルコードです。

ActivityのonCreateメソッド

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

これはActivityの最小コードです。開発者は「Activity」クラスの「setContentView」メソッドをよく目にしますが、ここまでの知識から「ContentView」は表示対象となる「View」を指定するためのメソッドであるとがわかります。「View」であれば、どんなものでも設定可能です。

サンプルコードでは「R.layout.activity_main」を指定してXMLから生成しています。「View」であればいいので次のように直接生成してもかまいません。

LayoutInflaterを利用してViewを生成する

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

        View v = getLayoutInflater()
                .inflate(R.layout.activity_main, null);
        setContentView(v);
    }
}
```

このサンプルコードは上記の「ActivityのonCreateメソッド」のようなスマートな書き方ではありません。「Layout」が「View」であることを示すための実証コードです。「Activity」クラスの「getLayoutInflater」メソッドは、XMLから「Layout（Layoutはすべてviewを親クラスに持つ）」を生成します。生成したViewを「setContentView」メソッドで指定しているため、前述の2つのリスト「ActivityのonC

reateメソッド」と「LayoutInflaterを利用してViewを生成する」の内容は同じ意味を持ちます。

「ViewGroup」のなかにViewを持つ構造を示すもうひとつのソースコードを、以下に例示します。

findViewByIdメソッド

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button b = (Button) findViewById(R.id.button);
        b.setOnClickListener(new OnClickListener() {
            ～省略～
        });
    }
}
```

このサンプルコードもよく見るソースコードです。「setContentView」メソッドでレイアウトを生成したあとに「findViewById」メソッドでボタンを取得しています。

現時点の知識をベースに解説すると「findViewById」メソッドは「Activityをルートとする『ViewGroup』のなかから指定のIDのViewを探してくる」という表現が可能です。

このことを理解しておくと、開発者がうっかりやってしまう、次のようなIDが見つからないエラーも、原因を説明できるようになります。

setContentViewメソッドより先にはfindViewByIdメソッドを利用できない

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Button b = (Button) findViewById(R.id.button);
        setContentView(R.layout.activity_main);

        b.setOnClickListener(new OnClickListener() {
            ～省略～
        });
    }
}
```

このようなケースでは「findViewById」メソッドは「null」を返します。「setContentView」メソッドによってレイアウトを生成する前のため「R.id.button」をもつ「View」が見つかりません。このまま実行すると次の通り「NullPointerException」が発生します。

NullPointerExceptionの内容

```
08-31 06:11:16.400: E/AndroidRuntime(914): FATAL EXCEPTION: main
08-31 06:11:16.400: E/AndroidRuntime(914): Process: com.example.ui.sample00, PID: 914
08-31 06:11:16.400: E/AndroidRuntime(914): java.lang.RuntimeException:
    Unable to start activity ComponentInfo{com.example.ui.sample00/
    com.example.ui.sample00.MainActivity}: java.lang.NullPointerException
```

この場合は「setContentView」メソッドのまえに「findViewById」メソッドを使い、期待する動作を得られなかったため「NullPointerException」が発生していました。

初心者にとっては、ViewのIDが見つからないことは、悩みの種となり得ますが、理由がわかれば迷わず、修正できます。

「View」と「ViewGroup」の関係を正しく把握することがUIを把握する手助けとなるはずです。

「findViewById」メソッドはActivityのみではなく「View」クラスにも存在しています。

11-1-4 Viewをカスタマイズする

Viewクラスを利用することで、独自機能を追加できます。カスタマイズのサンプルとして、Viewにペイント機能を追加してみましょう。サンプルアプリケーションは次の図のとおりです(**図3**)。

図3: ペイントアプリ

今回、作成するアプリでは「View」の機能を拡張します。タッチイベントを取得して、指の動きに合わせて線を描けるようにします。

CustomView.javaファイルを作成する

「View」クラスを継承した新しい「CustomView」クラスを作成していきます。Javaのファイルを新規に作成しましょう。「View」を継承するクラス名に応じてファイル名を決定します。

以下のサンプルコードでは「CustomView.java」で統一しています。新規でクラスを作る場合でも、「TextView」や「ImageView」のように、「～View」をつけることで、どのような役割をもっているのか一目でわかりやすくできます。

CustomView.java

```
package jp.androidopentextbook.ui.sample01_draw;

import android.content.Context;
import android.view.View;

class CustomView extends View {
    //コンストラクタ
    public CustomView(Context context) {
        super(context);
        // 何もしません
    }
```

現時点のサンプルコードでは、コンストラクタが1つですが「View」を作る際は、3つ用意します。

```
public View(Context context)
public View(Context context, AttributeSet attrs)
public View(Context context, AttributeSet attrs, int
defStyle)
```

それぞれのコンストラクタには意味があり、用途に応じて使い分けられます。XMLから参照される場合は、レイアウトパラメータである「AttributeSet」が必要です。「setContentView(R.layout.activity_main)」のように、XMLファイルから参照する場合は、前述のコンストラクタのうち2番目の「AttributeSet」を持つコンストラクタが呼び出されます。またスタイルを指定した場合は「defStyle」を持つ3つめのコンストラクタが使われます。

今回は「activity_main.xml」で定義するので次のように「public View(Context context, AttributeSet attrs)」を実装します。

CustomView.javaへのコンストラクタの実装

```
class CustomView extends View {

    //コンストラクタ
    public CustomView(Context context) {
        super(context);
        //何もしません
    }

    //スタイルを適用する際のコンストラクタ
    public CustomView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        //何もしません
    }

    //XMLより呼び出す際のコンストラクタ
    public CustomView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        setFocusable(true);
        initPaint();
    }

    // 描画用ビットマップ、キャンバス、パス、ペイント設定
    private Bitmap mBitmap;
    private Canvas mCanvas;
    private Path mPath;
    private Paint mPaint; //

    // 描画用Paintの初期化
    private void initPaint() {
    }
```

コンストラクタでは「setFocusable」メソッドを使ってフォーカスを有効にします。ここでは中身はありませんが、ペイント機能のための初期化メソッドもあらかじめ用意します。

レイアウトファイルに追加する

ここまでで最小限「View」として動作するようになりました。「activity_main.xml」には、次のように「CustomView」を追加します。

activity_main.xml

```
<RelativeLayout 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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <jp.androidopentextbook.ui.sample01_draw.CustomView
        android:id="@+id/paintView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

通常であれば「ImageView」であったり「TextView」「Button」などUIパーツ名であったりしますが、ここではパッケージ名を含めてクラス名を記載します。参照を間違えている場合「ClassNotFoundException」が発生します。その場合は「LogCat」に次のようなエラーログが出力されます。

エラーの内容

```
08-31 07:51:50.920: E/AndroidRuntime(855):
  Caused by: java.lang.ClassNotFoundException: Didn't find class
  "android.view.CustomView" on path: DexPathList[[zip file
  "/data/app/jp.androidopentextbook.ui.sample01_draw-1.apk"],
  nativeLibraryDirectories=[/data/app-lib/
  jp.androidopentextbook.ui.sample01_draw-1, /system/lib]]
```

エラーがある場合は「Didn't find class」に続き、「android.view.CustomView」など、誤ったクラス名が書かれているため、あらためて確認してください。

ペイント機能の実装

ペイント機能を実装するにあたって「Canvas」の用意と「Paint」などのリソースの初期化を行います。「Canvas」クラスは、描いた内容が消えないように描画する内容を保存します。「Paint」はどのような線を引くか、スタイルや色などリソースを保存するクラスです。

CustomView.java - 初期化と更新処理

```
class CustomView extends View {
    ～省略～

    // 描画用ビットマップ、キャンバス、パス、ペイント設定
    private Bitmap mBitmap;
    private Canvas mCanvas;
    private Path mPath;
    private Paint mPaint;

    // 描画用Paintの初期化
    private void initPaint() {
        mPath = new Path();

        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        mPaint.setDither(true);
        mPaint.setColor(Color.RED);
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setStrokeJoin(Paint.Join.ROUND);
        mPaint.setStrokeCap(Paint.Cap.ROUND);
        mPaint.setStrokeWidth(12);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        // 画面サイズ変更時の通知
        Log.v("View", "onSizeChanged Width:" + w + ",Height:" + h);

        // Canvasを作成
        mBitmap = Bitmap.createBitmap(w, h, Bitmap.Config.ARGB_8888);
        mCanvas = new Canvas(mBitmap);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        // Viewの描画関数で、パスを描画する
        canvas.drawBitmap(mBitmap, 0, 0, mPaint);
        canvas.drawPath(mPath, mPaint);
    }
    ～省略～
}
```

「initPaint」メソッドを変更して「Paint」と「Path」を初期化します。おなじ「initPaint」メソッド内で、線の種類や色、サイズなどを設定します。

「onSizeChanged」メソッドは「View」サイズが変更された場合に呼び出されるメソッドです。「View」のサイズが確定するタイミングでもあるので、サンプルでは「Bitmap」のサイズをここで決めています。

「onDraw」メソッドは「View」の描画メソッドです。Androidの「View」システムから呼び出され、「View」の表示内容を更新します。「Canvas」クラスの「drawPath」メソッドで指の動きを描画して画面へ反映します（指の動きは「CustomView」のメンバー変数「mPath」に保存しています）。

ペイント機能の基本操作となる描画部分は、タッチイベントを受け取る「onTouchEvent」メソッドです。以下のコードでは指がスクリーンに触れると描き始め、離れると描画を終了します。

CustomView.java - タッチイベント

```
class CustomView extends View {
    ～省略～
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        float x = event.getX();
        float y = event.getY();

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                touch_start(x, y);
                break;
            case MotionEvent.ACTION_MOVE:
                touch_move(x, y);
                break;
            case MotionEvent.ACTION_UP:
                touch_up();
                break;
        }
        invalidate();//Viewの再描画
        return true;
    }
    ～省略～
}
```

onDraw、onTouchEventなどメソッド名にonXXXがつくメソッドはAndroidのフレームワークが実行する前提であることがほとんどです。

「onTouchEvent」メソッドでは「ACTION_DOWN」と「ACTION_UP」をトリガーに、押下中状態(MOVE)で線を引く処理を実装しています。「invalidate」メソッドは、再描画を指示するメソッドです。「invalidate」メソッドを通じて「onDraw」メソッドが呼び出されます。サンプルでは簡単化のために座標に動きがある場合は必ず再描画しています。

最後にタッチ開始･移動･終了の内部処理です。次に解説するサンプルコードは、線を描画するために独自に追加したメソッドです。

CustomView.java - 描画処理

```
class CustomView extends View {
    ～省略～
    private float mX, mY;
    private static final float TOUCH_TOLERANCE = 4;

    private void touch_start(float x, float y) {
        //タッチ開始時
        Log.v("View", "touch_start");
        mPath.reset();
        mPath.moveTo(x, y);
        mX = x;
        mY = y;
    }

    private void touch_move(float x, float y) {
        //指が移動している間の処理
        Log.d("View", "touch_move");
        float dx = Math.abs(x - mX);
        float dy = Math.abs(y - mY);
        //閾値より移動量が多ければ線をつなぐ
        if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
            mPath.quadTo(mX, mY, (x + mX) / 2, (y + mY) / 2);
            mX = x;
            mY = y;
        }
    }

    private void touch_up() {
        //タッチ終了時
        Log.v("View", "touch_up");
        //線を描く
        mPath.lineTo(mX, mY);
        mCanvas.drawPath(mPath, mPaint);
        mPath.reset();
    }
    ～省略～
}
```

タッチによる描画のために、指の動きを「mPath」として保存しています。本来は、受け取った値をそのまま使えばよいのですが、タッチイベントは、誤差も含むため、X座標、Y座標の値を確認しつつ、移動量を補正して線を描いています。

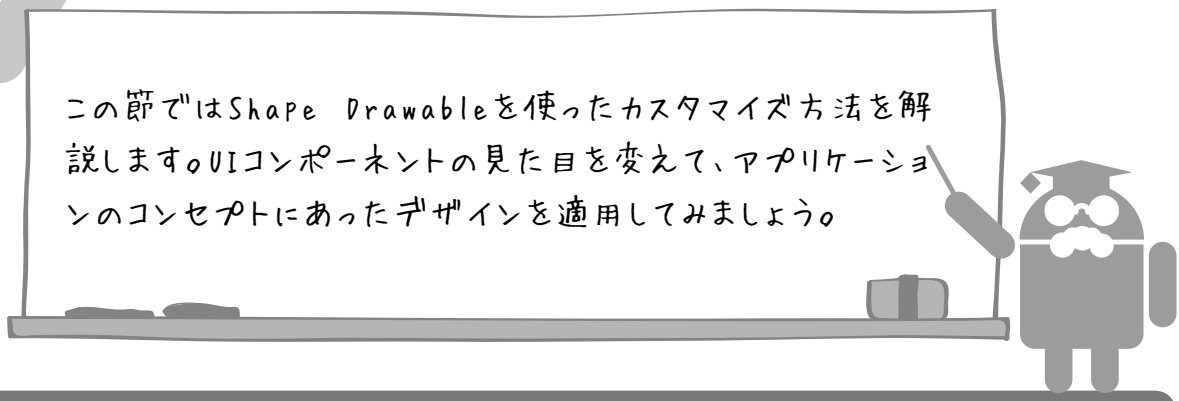
演習問題

(1)ペイント機能を拡張しよう

- **現在、利用できる色は赤のみですがボタンを追加して、緑、青でも絵を描けるようにしてみましょう**
- **余力があれば消しゴムや、Undo（戻る）機能を追加してみましょう**

11-2 UIコンポーネントの見た目をカスタマイズする

著：日高正博



この節で学ぶこと

- **UIコンポーネントのカスタマイズ**
- **Shape Drawableの使い方とリソースの生成**
- **状態を指定したリソース管理**

この節で出てくるキーワード一覧

UIコンポーネント
Shape Drawable
State List
ボタン
カスタマイズ
shape要素
corners要素
gradient要素
padding要素
size要素
stroke要素
selector要素

11-2-1 デザインを統一する

ここでは、UIコンポーネントの見た目を変更する手法について解説します。Androidの標準UIコンポーネントを利用することには、ユーザーが、アプリケーションごとの操作性の違いにと迷うことがなくなるという利点があります。新しくインストールしたアプリケーションであったとしても、見慣れたUIを操作できるわけです。

しかし、提供するサービスのブランドを考慮した結果、標準コンポーネントは使われないケースも増えています。多くのアプリケーションで、標準UIコンポーネントの見た目を変えて、デザイン性を高めるべくカスタマイズしています（**図1**、**2**）。

図1: Android標準の設定画面

図2: アプリケーション独自のカスタマイズ例（Wantedly）

Android標準の設定画面では、黒を基調とした落ち着いた見た目です。ところが、「Wantedly」アプリをみてみると、白を基調とした画面になっています。

ボタンなどUIコンポーネントは、現在主流のフラットデザインが採用されており、すっきりとまとめられています。

11-2-2 「Shape Drawable」を利用する

AndroidではXMLを使って図形を作成できます。簡単な図形であれば、形状に対していくつかの属性を指定することで、画像なしでも作成可能です。拡大縮小も行えるうえ、ファイルサイズを小さくできるという特徴があります。解像度が異なる場合も、ボケが生じたりすることがありません。これは「Shape Drawable」が「ベクターデータ」となっているからです。対してJPEGやPNGなどの画像は「ラスターデータ」と呼ぶことがあります。

Androidのリソースには、いくつかの種類がありますが「Shape Drawable」は名前のとおり「drawable」として扱います（**表1**）。

リソース	説 明
`animator/`	プロパティアニメーション
`anim/`	ビューアニメーション
`color/`	色の定義
`drawable/`	画像、描画リソース
`layout/`	レイアウトファイル
`menu/`	メニュー項目
`raw/`	ストリーム
`values/`	テキスト、スタイル
`xml/`	そのほかプログラムで利用するXML

表1：Androidのリソースの種類

「Shape Drawable」ファイルは次の場所に配置します。

```
res/drawable/filename.xml
```

このとき、filenameがリソースIDとして扱われます。Shape Drawableの文法は次のとおりです。

Shape Drawableの文法

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:useLevel=["true" | "false"] />
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
         android:bottom="integer" />
    <size
        android:width="integer"
        android:height="integer" />
    <solid
        android:color="color" />
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>
```

「Shape Drawable」では、四角、楕円、線、円の4つの形状から選択できます。「android:shape」属性の取り得る値は、**表2**のとおりです。

属 性	説 明
rectangle	四角形
oval	楕円形
line	線
ring	円

表2:android:shape属性値

「corners」要素では、四角形の四隅に対して半径を設定できます。角丸にする場合に利用します。「corners」要素は次の属性値を持ち、細かく角の半径を指定可能です(**表3**)。

属 性	説 明
android:radius	すべての角に対して半径を指定
android:topLeftRadius	左上の角半径
android:topRightRadius	右上の角半径
android:bottomLeftRadius	左下の角半径
android:bottomRightRadius	右下の角半径

表3:corners要素の属性値

「gradient」要素ではグラデーションを設定できます(**表4**)。

属 性	説 明
android:angle	グラデーション方向
android:centerX	中心位置(X座標)
android:center	中心位置(Y座標)
android:startColor	始端の色
android:endColor	終端の色
android:centerColor	中間色
android:gradientRadius	半径を指定
android:type	グラデーションのパターン
android:useLavel	LevelListDrawableを利用する

表4:gradient要素の属性値

「android:angle」属性値は、グラデーションの角度ですが0は左から右、90は下から上方向です。角度は、45の倍数でなければ機能しません。

「padding」要素では上下左右のパディングを設定できます(**表5**)。

属 性	説 明
android:left	左方向のパディング
android:top	上方向のパディング
android:right	右方向のパディング
android:bottom	右方向のパディング

表5:padding要素の属性値

「size」要素では大きさを設定できます(**表6**)。

属 性	説 明
android:height	高さ
android:width	横幅

表6:padding要素の属性値

「solid」要素では色を設定できます。使える色は1色のみです(**表7**)。

属 性	説 明
android:color	指定色

表7:solid要素の属性値

「stroke」要素で枠線の形状を設定できます(**表8**)。

属 性	説 明
android:width	枠線の太さ
android:color	枠線の色
android:dashGap	点線の間隔
android:dashWidth	点線の幅

表8:stroke要素の属性値

「Shape Drawable」では、こうした要素を使い分けて「Drawable」を作成します。また要素はそれぞれ単純な図形、ルールで構成されています。

複雑な形状は苦手なため、ある程度、シンプルに記号化したり、色の変化でバリエーションを持ったりして、アプリケーションのコンセプトに合わせます。

11-2-3 ボタンをカスタマイズする

「Shape Drawable」をつかってボタンを変更してみましょう。ここでは画像ではなく、XMLをつかってボタンの背景を作成します(**図3**)。

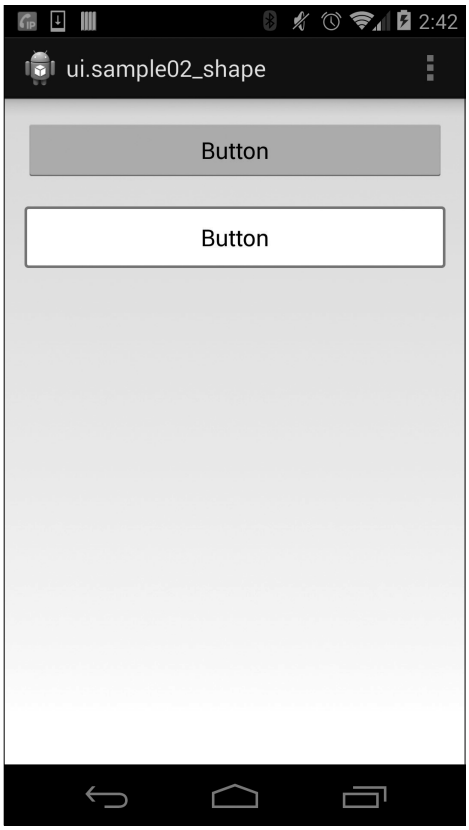


図3: 標準ボタンとカスタマイズしたボタン

State List: http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList

標準のUIコンポーネントを使っている場合、Android端末ごとのテーマやAndroidのバージョンごとに、統一されたデザインが適用されます。一方、ボタンをカスタマイズした場合は、常にアプリケーションの意図したデザインで変化はありません。

ボタンにはいくつかの状態があり、それらを管理するために「State List」という仕組みを利用します。以下で「State List」の文法を確認してみましょう。

State Listの文法

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android"
    android:constantSize=["true" | "false"]
    android:dither=["true" | "false"]
    android:variablePadding=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_hovered=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_activated=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

「State List」では「selector」要素と「item」要素が必要です。ボタンが押されている、フォーカスがあるなど、状態にあわせて表示する「Drawable」を切り替えます。(**表9**)

属 性	説 明
android:drawable	Drawableリソースの指定
android:state_pressed	押下
android:state_focused	フォーカス
android:state_hovered	ホバー
android:state_selected	選択状態
android:state_checkable	チェック可能
android:state_checked	チェック
android:state_enabled	有効／無効
android:state_activated	アクティベート
android:state_window_focused	Windowフォーカス

表9: State Listのitem要素の属性値

ボタンの場合「android:state_pressed」属性があれば、最低限の動作が可能です。実際には「android:state_pressed」「android:state_focused」「android:state_enabled」の3つを組み合わせる必要があります。以下、サンプルとし

て、ボタンの「State List」を作ってみます。

res/drawable/button_mycolor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- ボタンが押されたときの定義 -->
    <item android:state_pressed="true">
        ～省略～
    </item>

    <!-- ボタンが押されていないときの定義 -->
    <item android:state_pressed="false">
        ～省略～
    </item>
</selector>
```

省略した部分は「Shape Drawable」です（「Shape Drawable」については後述します）。「State List」は「selector」要素が「item」要素を持ち、それぞれの状態を定義します。

ボタンが押されたときの「Shape Drawable」は、次のとおりです。

res/drawable/button_mycolor.xml - ボタンが押されたとき

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- ボタンが押されたときの定義 -->
    <item android:state_pressed="true">
        <solid android:color="#ffffff" />
        <stroke android:color="#01579b" android:width="2dp" />
        <corners android:radius="2dp" />
        <padding android:left="10dp"
            android:top="10dp"
            android:right="10dp"
            android:bottom="10dp" />
        <size android:height="30dp"
            android:width="30dp" />
    </item>
    ～省略～
</selector>
```

ボタンが押されていないときの「Shape Drawable」は次のとおりです。

res/drawable/button_mycolor.xml - ボタンが押されていないとき

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
     ～省略～
    <!-- ボタンが押されていないときの定義 -->
    <item android:state_pressed="false">
        <shape android:shape="rectangle">
            <solid android:color="#ffffff" />
            <stroke android:color="#03a9f4" android:width="2dp" />
            <corners android:radius="2dp" />
            <padding android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp" />
            <size android:height="30dp"
                android:width="30dp" />
        </shape>
    </item>
</selector>
```

サンプルのボタンが押されたとき、押されていないときの違いは枠線「stroke」の色です。どちらも共通して背景色「solid」を白、丸角「corners」を2dp、パディングを10dp、サイズを30dpで設定しています。

大きく見た目が変わるとユーザーが驚くため、このサンプルコードでは意図的に変化を少なくしています。

ボタンを押した状態での見た目の変化には、ユーザーの操作に対するレスポンスにあたるため、注意が必要です。

このような場合は、ユーザーが想像できる範囲に「Shape Drawable」の変化をおさめると綺麗にみえます。枠線ではなく背景色を変えても、わかりやすいボタンになるでしょう。

作成したDrawable(「State List」を使ったオリジナルのボタン)を利用するにはレイアウトを次のように変更します。

res/layout/activity_main.xml

```
<RelativeLayout 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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
```

```
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="20dp"
        android:text="Button" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button1"
        android:background="@drawable/button_mycolor"
        android:layout_marginBottom="20dp"
        android:text="Button" />

</RelativeLayout>
```

ボタンの背景（「android:background」属性）に作成した「Drawable」を指定すると、見た目に反映されます。サンプルアプリを動かして確認してみましょう。

また、応用として丸いボタンを作る場合のサンプルコードを以下に示します。

res/drawable/button_oval.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true">
        <shape android:shape="oval">
            <solid android:color="#01579b" />
        </shape>
    </item>

    <item android:state_pressed="false">
        <shape android:shape="oval">
            <solid android:color="#03a9f4" />
        </shape>
    </item>
</selector>
```

状態にあわせて「Shape Drawable」の背景を変更しています。ここでは「android:color」属性値を直値で入力していますが「color.xml」に定義する方法も利用できます。

作ったUIコンポーネントを修正しないケースは、まずありません。出来上がったパーツを少し変更したい、色味を確認したいなど、細かな修正が必ず発生します。その際、リソースを分割して管理すると変更箇所が少なくなり効率があがります。

「button_mycolor.xml」のもうひとつの解として、次のように書くこともできます。

res/drawable/button_mycolor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- ボタンが押されたときの定義 -->
    <item android:state_pressed="true">
        <shape android:shape="rectangle">
            <stroke android:color="#01579b" android:width="2dp" />
        </shape>
    </item>

    <!-- ボタンが押されていないときの定義 -->
    <item android:state_pressed="false">
        <shape android:shape="rectangle">
            <stroke android:color="#03a9f4" android:width="2dp" />
        </shape>
    </item>

    <item>
        <shape android:shape="rectangle">
            <solid android:color="#ffffff" />
            <stroke android:color="#01579b" android:width="2dp" />
            <corners android:radius="2dp" />
            <padding android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp" />
            <size android:height="30dp"
                android:width="30dp" />
        </shape>
    </item>
</selector>
```

ここでは差分のある「stroke」要素のみを取り出して定義しています。共通部分の属性を持たない「item」要素にまとめて記述することでXMLのコードの記述量を減らせます。

演習問題

(1)モックアップを作ってみよう

モックアップとは、アプリケーションを作る際にレイアウト、操作感を検討するための試作です。

ここでは与えられたテーマに沿ってモックアップを作ってみましょう。

・次の表現からテーマカラーを決めてみましょう

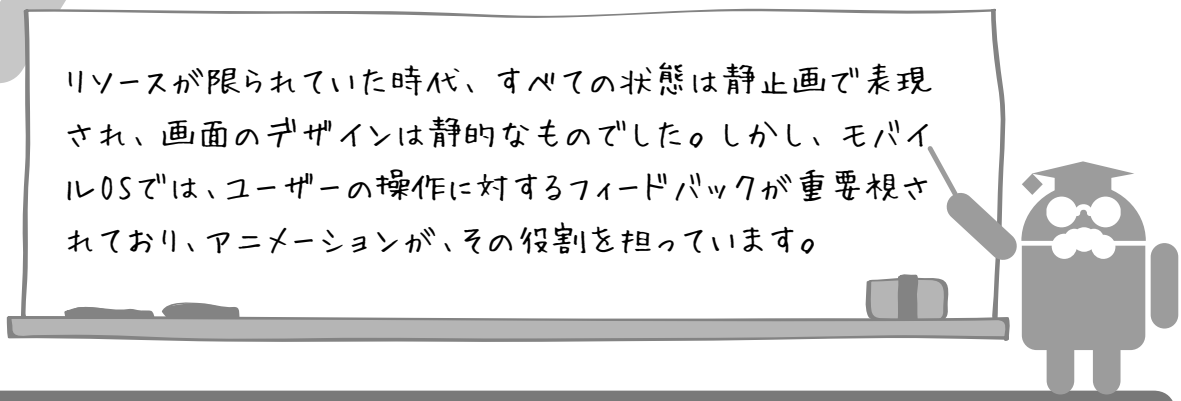
「アクティブ」「落ち着いた」「クール」「優しい」「温かい」、または自分で作りたいアプリのコンセプトをきめる

・次の内容のアプリケーションでレイアウトを考えてみましょう

「電卓」「フォト」「ミュージック」「ニュース」「電話」「時計」

11-3 アニメーション

著：日高正博



この節で学ぶこと

- Viewにアニメーションを設定する
- アニメーション効果を理解する
- アニメーションを任意のタイミングで利用する

この節で出てくるキーワード一覧

Frameアニメーション
アニメーションの繰り返し
Tweenアニメーション
Viewの切り替え、移動、回転、拡大、縮小
Animation
AnimationDrawable
animation-list要素
onWindowFocusChanged
set要素
alpha要素
scale要素
translate要素
rotate要素
View
ImageView

11-3-1 アニメーションの種類

アニメーションのタイプは2つに大別できます。「Frameアニメーション（AnimationDrawable）」と「Tweenアニメーション（Animation）」です。それぞれ次のような特徴があります。

Frameアニメーションは、順番にイメージを並べて表示してアニメーションにします。順番に表示させる画像をあらかじめ用意する、パラパラ漫画のようなものです。

Tweenアニメーションは、1つのイメージを連続で変化させます。具体的には、物体の移動やフェード、回転、拡大縮小といった効果を組み合わせて変化させる

ことができます。それぞれについて、詳しく見ていきます。

11-3-2 Frameアニメーション

「Frameアニメーション（AnimationDrawable）」を使うと、XMLでリソースを用意するだけで簡単にアニメーションが作成できます。時間ごとに切り替えてアニメーションを実現するパラパラ漫画のような手法です。今回は次の画像を事前に用意して、回転アニメーションを実現します（**図1**）。

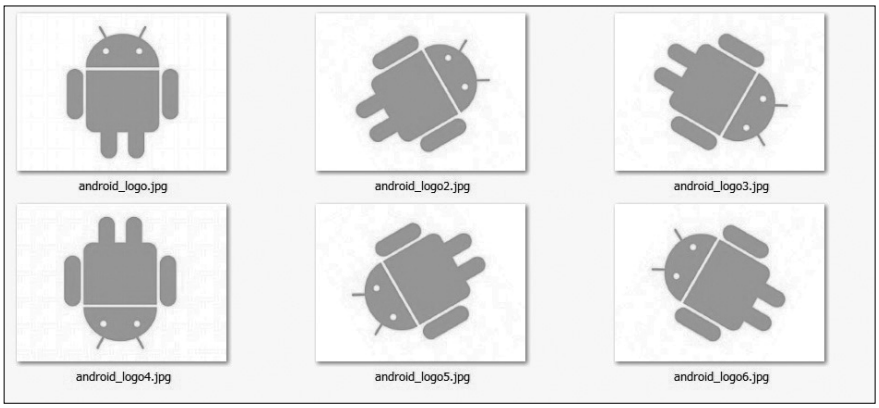


図1: 用意する画像一覧

今回は6枚の画像を連続表示して回転しているように見えるアニメーションを作成します（**図2**）。

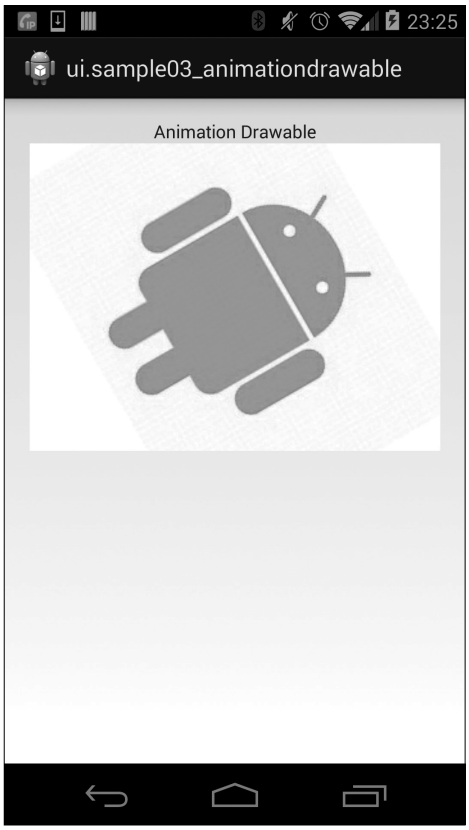


図2: Frameアニメーションのサンプル

「Frameアニメーション」に使うリソースファイルの保存場所は次のとおりです。

```
res/drawable/filename.xml
```

今回はXMLでアニメーションの設定を行うのでdrawableの下にXMLも一緒に配置します（**図3**）。

図3: リソースの配置位置

新規に作成するXMLファイル名は「android_animation.xml」です。

android_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
     android:oneshot="false">
    <item android:drawable="@drawable/android_logo" android:duration="50" />
    <item android:drawable="@drawable/android_logo2" android:duration="50" />
    <item android:drawable="@drawable/android_logo3" android:duration="50" />
    <item android:drawable="@drawable/android_logo4" android:duration="50" />
    <item android:drawable="@drawable/android_logo5" android:duration="50" />
    <item android:drawable="@drawable/android_logo6" android:duration="50" />
 </animation-list>
```

「animation-list」要素が「item」要素を持ち、1枚の「Frame」を表します。6枚分をすべて登録します。「android:duration」属性はミリ秒（ms単位）で指定できます。サンプルでは50ミリ秒です。「android:oneshot」属性が「false」のときは、リピート再生を意味します。「true」なら1度のみアニメーションを行います。

「animation-list」の持つ属性は**表1**のとおりです。

属 性	説 明
android:drawable	drawableリソースへの参照（1フレーム分）
android:duration	アニメーションの間隔（ms単位）
android:oneshot	繰り返し設定
android:variablePadding	パディングを可変にする
android:visible	初期表示状態

表1：animation-list要素の属性一覧

アニメーションを開始するためのサンプルコードは次のとおりです。実行のためには、事前にレイアウトファイル「activity_main.xml」に「ImageView」を用意してください。

MainActivity.java

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
     super.onWindowFocusChanged(hasFocus);
     ImageView img = (ImageView)findViewById(R.id.imageView);

     // AnimationDrawableのXMLリソースを指定
     img.setBackgroundResource(R.drawable.android_animation);
     // AnimationDrawableを取得
     AnimationDrawable frameAnimation = (AnimationDrawable) img.getBackground();
     // アニメーションの開始
     frameAnimation.start();
    }
}
```

「AnimationDrawable」を「ImageView」の背景に設定しています。注意点は「AnimationDrawable」クラスの「start」メソッドは「onCreate」メソッドが呼ばれた時点では有効になっていないことです。「onCreate」メソッドで呼び出してもアニメーションしません。画面表示の準備が完了するまでは無視されるためです。このサンプルのように「onWindowFocusChanged」メソッド（画面描画の準備が完了した段階）で呼び出します。

11-3-3 Tweenアニメーション

ひとつの「View」を連続的に変化させる「Tweenアニメーション」について解説します。アニメーションの動きはXMLとソースコードのどちらでも定義できますが、多くの場合、アニメーションがプログラマブル（可変）である必要はありません。XMLで静的に保持したほうがメンテナンスしやすいでしょう。ここでは「View」に対して透明な状態からフェードインする「Tweenアニメーション」を考えます（**図4**）。

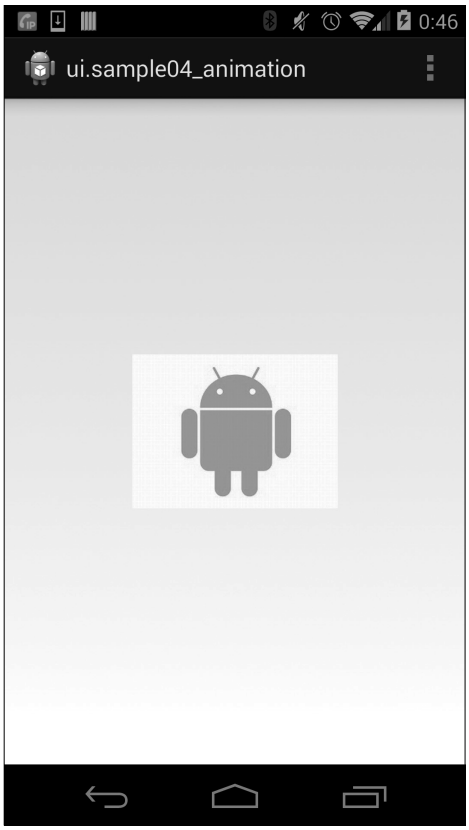


図4: サンプルアプリ画面

「Tweenアニメーション」のXMLファイルは、次のとおり「anim」以下に配置します。

文法の詳細は以下を参照してください（ http://developer.android.com/guide/topics/resources/animation-resource.html#Tween）

```
res/anim/filename.xml
```

文法は次のとおりです。

Tweenアニメーションの文法

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"] >
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>
```

「Tweenアニメーション」をXMLファイルで記述する場合「set」「alpha」「scale」「translate」「rotate」いずれかの要素を持っている必要があります。

「set」要素やその他のアニメーション要素に使える属性は**表2**のとおりです。

属 性	説 明
android:detachWallpaper	壁紙のアニメーションを停止する
android:duration	実行時間を指定
android:fillAfter	アニメーションが終わった後の状態を保つ
android:fillBefore	アニメーションが始まる前の状態を保つ
android:fillEnabled	fillAfter、fillBeforeを有効化する
android:interpolator	動きをスムーズに補完する
android:repeatCount	繰り返し回数
android:repeatMode	繰り返し動作設定（標準はループ）
android:startOffset	開始までのオフセット時間
android:zAdjustment	コンテンツ順序を奥行きで調整

表2:objectAnimator要素の属性一覧

こうした属性値は「set」「alpha」「scale」「translate」「rotate」いずれの要素でも有効です。

「alpha」要素の属性は**表3**のとおりです。「alpha」要素は、透明度を変更する要素です。

属 性	説 明
android:fromAlpha	開始時の透明度
android:toAlpha	終了時の透明度

表3:alpha要素の属性一覧

「scale」要素の属性は**表4**のとおりです。「scale」要素は「View」の拡大・縮小を行う要素です。

属 性	説 明
android:fromXScale	開始時の拡大率
android:toXScale	終了時の拡大率
android:fromYScale	開始時の拡大率
android:toYScale	拡大の中心となる座標
android:pivotX	右下の角半径
android:pivotY	拡大の中心となる座標

表4:scale要素の属性一覧

「translate」要素の属性は**表5**のとおりです。「translate」要素は、「View」を水平または垂直方向へ移動する要素です。

属 性	説 明
android:fromXDelta	開始時のオフセット
android:toXDelta	終了時のオフセット
android:fromYDelta	開始時のオフセット
android:toYDelta	終了時のオフセット

表5:translate要素の属性一覧

「rotate」要素の属性は**表6**のとおりです。「rotate」要素は「View」を回転する要素です。

属 性	説 明
android:fromDegrees	開始時のオフセット
android:toDegrees	終了時のオフセット
android:pivotX	回転の中心となる座標
android:pivotY	回転の中心となる座標

表6:rotate要素の属性一覧

「View」に対してアニメーションを設定する場合は、こうした要素や属性を利用して以下のようにアニメーション用のXMLを作成します。

res/anim/motion.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:duration="5000">
    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0" />
</set>
```

サンプルコードでは、透明な状態からフェードインして「View」が表示されます。「set」要素でアニメーションにかかる時間を5秒と設定しているため、とてもゆっくりと画面に表示されます。

作成したアニメーションXMLファイルはそのままでは適用されないため、ソースコード上で設定が必要です。

MainActivity.java

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ImageView img = (ImageView) findViewById(R.id.imageView);
        //Tweenアニメーションの適用
        Animation animation= AnimationUtils.
                loadAnimation(this,R.anim.motion);
        img.startAnimation(animation);
    }
}
```

「ImageView」に対して「startAnimation」メソッドを使って「Tweenアニメーション」を設定しています。「Tweenアニメーション」そのものは「AnimationUtils」クラスの「loadAnimation」メソッドを使って、リソースファイル「motion.xml」から生成しています。

アニメーションは、「View（レイアウトを含む）」の表示時や終了時などユーザー操作に応じて利用してください。画面内で勝手にアニメーションしたり、アニメーションの目的を感じない無意味な動きをしたりすると逆効果です。アニメーションをたくさん使用するより、注目させたいアクションまたは注目してほしいタイミングに絞って使うといいでしょう。

演習問題

(1)サンプルアプリにアニメーションを適用してみましょう。

アニメーションの組み合わせは多く、用途に応じて作る必要があります。

代表的な表示方法として透明度を変更するフェードイン、位置を変えるスライドイン、拡大率を変えるズームインなどがあります。

- **上記の効果を持つアニメーションを作成して効果を確認してましょう**
- **複数の「View」「ImageView」を用意して同時にアニメーションさせてみよう**