



関連記事

2014-06-19

## Androidの独自レイアウトを作成する

Category Android

### 概要

以下のように、layoutのxmlを設定するだけで、自動的に意図するレイアウトになるような独自レイアウトを作成する。

```
<com.example.tile.TileLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
xmlns:custom="http://schemas.android.com/apk/res/com.example.tile"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity" >

<com.example.tile.TileItemView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="1dp"
    android:contentDescription="@string/metallic"
    custom:src="@drawable/image1"
    custom:subtile="@string/android23"
    custom:title="@string/metallic" />

<com.example.tile.TileItemView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="1dp"
    android:contentDescription="@string/honeycomb"
    custom:src="@drawable/image2"
    custom:subtile="@string/android30"
    custom:title="@string/honeycomb" />

<com.example.tile.TileItemView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="1dp"
    android:contentDescription="@string/normal"
    custom:src="@drawable/image3"
    custom:subtile="@string/android10"
    custom:title="@string/normal" />

<com.example.tile.TileItemView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="1dp"
    android:contentDescription="@string/gopher"
    custom:src="@drawable/image4"
    custom:subtile="@string/golang"
    custom:title="@string/gopher" />

</com.example.tile.TileLayout>
```



### 詳細

子Viewをもち、その配置を決めるにはViewGroupを継承する必要がある。  
ViewGroupを継承したクラスはonMeasure(),onLayout()メソッドを実装する。  
onMeasureでは子Viewの大きさを、onLayoutでは子Viewの位置を指定する。

```
public class TileLayout extends ViewGroup {

    public TileLayout(Context context) {
        this(context, null);
    }

    public TileLayout(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public TileLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    // 子Viewの大きさを決定
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        //
    }

    // 子Viewの位置を決定
    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        //
    }
}
```

ビューを描画する際、どのサイズで、どこに描画するか決定している必要がある。  
Androidでは以下の手順でサイズと位置を決めている。

ルートビューからビュー階層をたどって、すべての子ビューのサイズを決定する(onMeasure)  
ルートビューからビュー階層をたどって、すべての子ビューを配置する(onLayout)

onMeasureでは親のViewGroupによって割り当てられたサイズとレイアウトモードが渡される  
これをもとに実際のサイズを計算して、最終的なサイズをsetMeasureDimension()でセットする  
レイアウトモードには以下の三つがある

- MeasureSpec.EXACTLY...親のViewGroupから大きさを指定したい場合(match\_parentとか固定値とか)
- MeasureSpec.AT\_MOST...子View自身で大きさを決定したい場合 (wrap\_contentとか)
- MeasureSpec.UNSPECIFIED...割り当てられたサイズ内に収まる必要サイズを計算、親のViewGroupのサイズに関係なく、子Viewが任意のサイズになっている場合はUNSPECIFIEDにする

今回のレイアウトでは親レイアウトから大きさを指定したいのでEXACTLY以外のときはエラーとする。  
またonMeasureではsetMeasureDimension()を呼んで自分（親）のサイズを指定する必要があるので、指定されているレイアウトのサイズのままsetMeasureDimension()を呼ぶ。

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    // このViewGroupに指定されているレイアウトのモードを取得する
    final int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    final int heightMode = MeasureSpec.getMode(heightMeasureSpec);

    if (widthMode != MeasureSpec.EXACTLY || heightMode != MeasureSpec.EXACTLY) {
        // レイアウトモードがEXACTLY以外のときはエラーにする
        throw new IllegalStateException("Must measure with an exact width");
    }

    // このViewGroupに割り当てられているサイズを取得する
    final int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    final int heightSize = MeasureSpec.getSize(heightMeasureSpec);

    // このViewGroupのサイズをセットする
    setMeasuredDimension(widthSize, heightSize);
}
```

次に子Viewの設定。  
まずはサイズの指定。

```
// padding分を差し引いて親の幅と高さを求める
int width = widthSize - getPaddingLeft() - getPaddingRight();
int height = heightSize - getPaddingTop() - getPaddingBottom();

int childWidth1;
int childHeight1;

int childWidth2;
int childHeight2;

int childWidth3;
int childHeight3;

// 2番目の子ビューの幅 = 高さ = 親の幅 * 2 / 3
childWidth2 = childHeight2 = width * 2 / 3;

// 1番目の子ビューの幅 = 親の幅
childWidth1 = width;
// 1番目の子ビューの高さ = 親の高さ - 2番目の子ビューの幅
childHeight1 = height - childHeight2;

// 3,4番目の子ビューの幅 = 親の幅 - 2番目の子ビューの幅
childWidth3 = width - childWidth2;
// 3,4番目の子ビューの高さ = 2番目の子ビューの高さ / 2
childHeight3 = childHeight2 / 2;
```

すべての子Viewに大して、大きさを設定。

```
// ViewGroup配下のすべての子Viewの数取得
final int childCount = getChildCount();

for (int i = 0; i < childCount; i++) {
    View child = getChildAt(i);
    TileLayout.LayoutParams params = (LayoutParams) child
        .getLayoutParams();

    int childWidth = 0;
    int childHeight = 0;
    if (i == 0) {
        // 1番目の子ビュー
        childWidth = childWidth1 - params.leftMargin
            - params.rightMargin;
        childHeight = childHeight1 - params.topMargin
            - params.bottomMargin;
    } else if (i == 1) {
        // 2番目の子ビュー
        childWidth = childWidth2 - params.leftMargin
            - params.rightMargin;
        childHeight = childHeight2 - params.topMargin
            - params.bottomMargin;
    } else if (i < 4) {
        // 3,4番目の子ビュー
        childWidth = childWidth3 - params.leftMargin
            - params.rightMargin;
        childHeight = childHeight3 - params.topMargin
            - params.bottomMargin;
    } else {
        // 5番目以降の子ビューのサイズは0（無視する）
        childWidth = 0;
        childHeight = 0;
    }
}
```

子Viewに大してmeasure()を実行。

```
// 子ビューに対してmeasure()を呼んでサイズを指定する
// TileLayoutでは子ビューのlayout_heightやlayout_widthの指定に関係なく
// 決まったサイズで配置するのでMeasureSpec.EXACTLYにする
int childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(childWidth,
    MeasureSpec.EXACTLY);
int childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(
    childHeight, MeasureSpec.EXACTLY);
child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
}
```

次にonLayout()の実装  
各子Viewの画面サイズ取得。

```
@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    // padding分を差し引いて子ビュー用の領域を求める
    int width = (r - l) - getPaddingLeft() - getPaddingRight();
    int height = (b - t) - getPaddingTop() - getPaddingBottom();

    int left = getPaddingLeft();
    int top = getPaddingTop();
    int right = left + width;
    int bottom = top + height;

    // 1つめは垂直幅、2つめ以降は正方形
    // 2つめは縦幅の2/3
    int childWidth2 = width * 2 / 3;
    // 1つめは画面高さから、二つ目の高さを引いただけの高さ
    int childHeight1 = height - childWidth2;
    // 3, 4つめは2つめの高さの半分
    int childHeight3 = childWidth2 / 2;
```

layoutを使って各子Viewを配置

```
int childCount = getChildCount();
for (int i = 0; i < childCount; i++) {
    if (i > 3) {
        // 5番目以降の子ビューは配置しない（無視する）
        break;
    }

    View child = getChildAt(i);

    switch (i) {
        case 0: {
            // 1番目の子ビュー
            childLayout(child, left, top, right, top + childHeight1);
            break;
        }
        case 1: {
            // 2番目の子ビュー
            childLayout(child, left, top + childHeight1,
                left + childWidth2, bottom);
            break;
        }
        case 2: {
            // 3番目のビュー
            childLayout(child, left + childWidth2, top + childHeight1,
                right, top + childHeight1 + childHeight3);
            break;
        }
        case 3: {
            // 4番目のビュー
            childLayout(child, left + childWidth2, top + childHeight1
                + childHeight3, right, bottom);
            break;
        }
    }
}

private void childLayout(View child, int l, int t, int r, int b) {
    // マージンだけずらして配置する
    TileLayout.LayoutParams params = (LayoutParams) child.getLayoutParams();
    child.layout(l + params.leftMargin, t + params.topMargin, r
        - params.rightMargin, b - params.bottomMargin);
}

public static class LayoutParams extends ViewGroup.MarginLayoutParams {

    /**
     * {@inheritDoc}
     */
    public LayoutParams(Context c, AttributeSet attrs) {
        super(c, attrs);
    }

    /**
     * {@inheritDoc}
     */
    public LayoutParams(int width, int height) {
        super(width, height);
    }

    /**
     * {@inheritDoc}
     */
    public LayoutParams(ViewGroup.LayoutParams source) {
        super(source);
    }

    /**
     * {@inheritDoc}
     */
    public LayoutParams(ViewGroup.MarginLayoutParams source) {
        super(source);
    }

    public LayoutParams(LayoutParams source) {
        super(source);
    }
}
```

参考

ViewGroup直下にある全ての子Viewにアクセスするには I GE Android Blog  
http://www.sakajp.blog/archives/29829

JavaDoc 1.4の新機能 - (@inheritDoc) - mars のメモ



Android Pattern Cookbook マークシートで埋もれないための差別化戦略  
作者: あんざいゆき  
出版社: かんきょう出版  
発売日: 2014/03/20  
メディア: 単行本（ソフトカバー）  
この商品を含むブログ (2件) を見る

うさがに (jditiro105) 5年前

★+

6

0

ツイート

B!ブックマーク

シェア



はてなブログをはじめよう！

tirot105さんは、はてなブログを使っています。あなたもはてなブログをはじめてみませんか？

はてなブログをはじめ（無料）

はてなブログとは