梨好きプログラマの人生ログ

20代後半にプログラマに転身。スマホアプリ開発したり、Webを勉強したり。 技術関連メモ、本の紹介、その他趣味など。

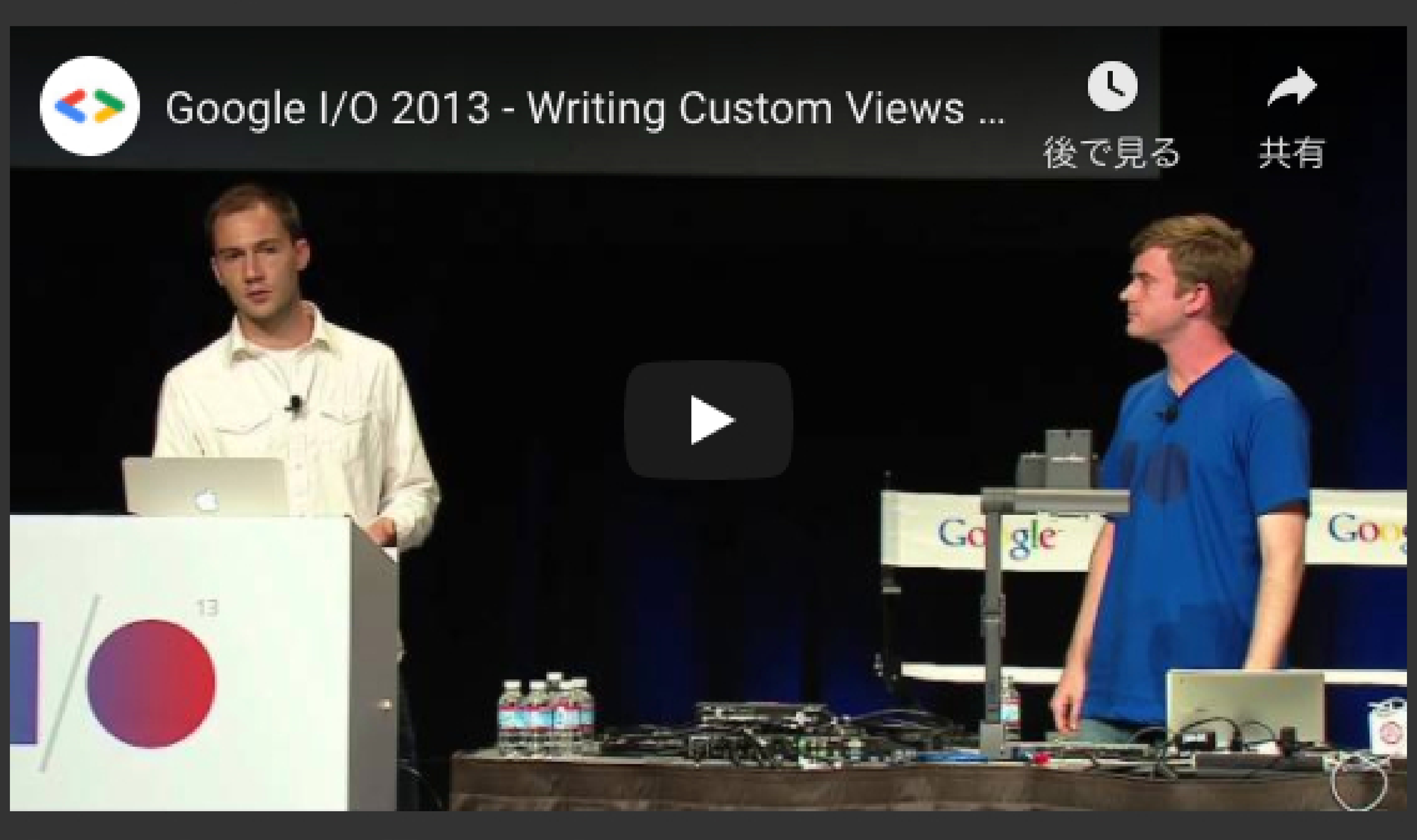
2014年7月25日金曜日

[Android] ViewのonMeasureとかonLayoutについてまとめてみる

[Android] ViewのonMeasureとかonLayoutについてまとめてみる

Androidアプリを作るとき、よくCustomLayout, CustomViewを作ります。
しかしなんとなく作りたいものが作れるけど、onMeasureとかonLayoutとかよく分からないまま使っていました。
折角なので、View周りの仕組みについて細かく調べてみた。

参考にしたのは、Google I/O 2013のWriting Custom Views for Android。それとAndroid DevelopersのReferenceから。
まずは本家Googleさんの説明を学んでみたいと思います。



Attachment/Detachement

onAttachedToWindow

YouTubeの４分前後から始まります。Android Developersはココのページ

Referenceだと下記のような説明になっています。

> *This is called when the view is attached to a window. At this point it has a Surface and will start drawing. Note that this function is guaranteed to be called before onDraw(android.graphics.Canvas), however it may be called any time before the first onDraw – including before or after onMeasure(int, int).*
>
> 訳：ビューがウィンドウにアタッチされた時に呼ばれる。このときビューはSurfaceを持っており、描画が始まる。このメソッドはonDrawの前に呼ばれることを保証しているが、最初のonDrawの前に呼ばれることがある。onMeasureの前後も含まれる。

Google I/Oでは、こんな感じ

- addViewとかした時に親ビューがWindowにアタッチされていれば、まず呼ばれるのがこのメソッド
- View自身がアクティブになるのを知るときがこのタイミング
- いろんなlistenerを登録するのはこの時
  - ListViewでは、このタイミングでデータ変更のListenerをRegisterしている

onDetachedFromWindow

referenceはココ。
Google I/Oでは、こちらの方が重要だと言ってますね。
例のごとく、Referenceから引用します。

> *This is called when the view is detached from a window. At this point it no longer has a surface for drawing.*
>
> 訳：ビューがウィンドウから切り離されたときに呼び出される。この時点では、描画処理はもう走らない。

- ビューがremoveされた時に呼び出される
  - ListViewをスクロールしてセルビューがrecycleされる時、removeViewしたとき、Activityを破棄しているときなど
- このタイミングはいろいろとスケジュールされていたものを終わらせる最適なタイミングである
  - 例えばアニメーション
  - バックグラウンドで走っていたスレッド
  - はたまたdrawing cacheなど、ここで抹消するのが最適であります
  - 特にAndroid 3.0から追加されたHardware Renderer、これはWindowにアタッチされたタイミングで数々のネイティブオブジェクトを生成している。なので、onDetachedFromWindowでクリーンナップする必要がある。なぜかって？コレ以降はViewはそれらを使う必要がいっさいないからさ。的な。
- また、デタッチされた段階でいろいろクリーンされるので、これらのメソッドを呼ぶ場合は、十分に気をつける事。いろんなバグが出るかもしれない。
- なぜ、有名なAvtivityのライフサイクルのようにViewのライフサイクルというものが存在しないのか
  - Activityはアプリケーションのトップレベルにいるものだが、Viewはそれよりも低い所にいる概念である。そのためViewはActivityのライフサイクルなんか知ったこっちゃない。
  - 自分でCustomViewを作り、listener/callbackを自分で実装しよう。そのcallback達をinvokeするのがActivityの役目である。

Traversal

YouTubeの9分20秒くらいからです。
Traversal?って何だって感じですが、直訳すると「横断」。よく分かりませんが、この場合、親ビュー子ビューを横切って処理する事を意味しているのかと思われます。

そんなTraversalのトリガーになるのは３つある。
1. animation
2. requestLayout()
3. invalidate()
気になるならViewRootImpl.javaを見ましょう。GrepCodeとかであるでしょうね。

- traversalのリクエストが出されたら、measurement、layout、drawingの順番に処理されて行く。
  - この順番は保証されている。
- 複雑なのはmeasurementとlayout。これらは常に一緒についてまわる。君のアプリで変なことをやっていない限りは。

Measurement And Layout

YouTubeの10分50秒くらいから。

- ViewGroupは子ビューのMeasure、Layoutを必ず行う。
- requestLayout()がすべてのステップの始まりとなる。
  - これによってすべてのビューの再計測が始まる。
- requestLayout()は再帰的な処理を行う。サイズの変更は関連するツリー構造をたどってすべてのビューに関して処理を行う。
  - これが、レイアウトに関してあまり深い階層で組まない方がよい理由でもある。
  - 深い階層の場合、requestLayout()を呼ぶと再帰的に処理が進んでしまうため、場合によってはすごい時間がかかってしまう。この事態を避けたいので、GoogleとしてもUI Toolkitで警告するようにはしているけども。

onMeasure()

- Viewそれ自身のサイズ及び、すべての子ビューのサイズを決定する。
- Viewのmeasure()メソッドにより呼び出される。
  - 再帰的な処理のため、すべての子ビューにわたってmeasure()が呼び出され、ビューのサイズが決定するのである。
- 計測するためのパラメータはMeasureSpecでまとめられている。
  - makeMeasureSpec()メソッドによって、パラメータがひとまとめにされ、それぞれのgetterメソッドにより取得する事が可能。
- ViewGroupのサイズが計測されると、ViewGroupは子ビューのサイズも計測する責任がある。
- onMeasure()は実際にはreturnがない。なので、setMeasuredDimension()を明示的に呼んで、widthとheightをセットする必要がある。これ忘れたら、frameworkはexceptionを投げますよ。
- ここで設定したwidth, heightはgetMeasuredWidth(), getMeasuredHeight()で取得できる。
  - getWidth(), getHeight()ではないので気をつけましょう。
  - これらのメソッドは、レイアウトが完了するまで値を持たないのですよ。

MeasureSpec

YouTubeの14分後半から。
3つ種類アルヨ。

MeasureSpec.EXACTLY

- そのまんま。固定的な値の時に使われる。
  - match_parent, OOOdp, layout_weighty=1など。

MeasureSpec.AT_MOST

- MeasureSpecで設定された値に等しい、もしくはそれ以下のサイズになるよ、ということ。
  - 親Viewのサイズがwrap_contentの場合、子ビューのサイズを決定するのにAT_MOSTが使われる。

MeasureSpec.UNSPECIFIED

- どんなサイズになるか分かりません。
- ビューは可能な限り無限にサイズを持つ。
- ScrollView, ListViewなんか典型的な例ですね。

getChildMeasureSpec()

適切なMeasureを得るためのフレームワークの一つがこれ。

- 例
  - parentのレイアウトのサイズが300pxの場合、で子ビューがwrap_contentのとき。
    - parentビューが小ビューに送る情報は、AT_MOSTモードで最大300px

onLayout()

YouTubeの17分くらいから。

- Measurementが完了したら、次は子ビューたちのposition(位置)を決定させる。
- そう、この時によくあるoffset, margin, padding等が適用されるのである。
- traversalの処理の中で一回呼ばれるのがこのメソッド。
  - measureの段階だと、何回も呼ばれてアレなので、重い処理などはここで行うのが適切である。
- サイズに関しては、onMeasureで既に計算がされている。なので、サイズを取得したい時はここでgetMeasuredWidth(), getMeasuredHeight()を呼ぶといいよ。

LayoutParams

YouTubeの20分くらいから。
XMLの属性でよく定義されるLayoutParamsですが、layoutに関連しているstaticなクラスです。

- 一番基本的なクラスはViewGroup.LayoutParams。
- その他にもLinearLayout.LayoutParamsやFrameLayout.LayoutParamsなどもある
  - これらを使わなくても基本的なLayoutParamsを使っていればいいよ。
- その他に便利なものがMarginLayoutParams。
- LayoutParamsはXMLからもインポートする事ができる。
- レイアウトに設定できるもので、widthやheightはおなじみですが、LinearLayoutの場合はweightというものもある。
- withやheightの他に必要があってCustom LayoutParamsを作ったなら、４つのメソッドを使う事ができるよ。
  - checkLayoutParams:
    - ViewにAddした時に呼び出される
    - そのLayoutParamsがViewGroupのものと合致しているかをチェックしている。合っていないとExceptionを投げちゃうよ。
    - LayoutをAddしたとき、前のViewがFrameLayoutだったときはFrameLayout.LayoutParamsが正解
  - generateLayoutParams(LayoutParams)
  - generateLayoutParams(AttributeSet)
  - generateDefaultLayoutParams
    - これらはReference参照
- onMeasureとかonLayoutで重い処理をやっているなら、LayoutParams使ってみるといいですよ。RelativeLayout.javaなんかすごい参考になる。

Draw

YouTubeの22分40秒くらいから。

- invalidate()によって呼び出される。
  - invalidateChild()が再帰的に親ビューに向かって呼び出される。
- これは非常に重い処理である。requestLayout()は単にフラグを処理しているだけだが、invalidate()はアニメーションや様々な描画処理が走る。
- onDraw()はOverrideすることができる。
- setWillNotDraw()というメソッドがあり、引数にtrueを指定すると描画処理をスキップする事ができる。子ビューは描画されるが。これを指定するとonDraw()が呼ばれずに、代わりにdispatchDraw()というものが呼ばれるよ。親ビューの描画処理はされずに、子ビューたちの描画を行うようになる。
- drawChild()メソッドによって、トランスフォーム、アニメーション等の描画処理が子ビューに対して処理される。これもOverrideできるので、すべての子ビューに対して呼んであげると良い

なんか非常に長くなりました。
最後なんかもう疲れて飛ばしまくりましたが、
とりあえず、Androidのソースコード読むといろいろわかりそうですね。
またそのうち、レポートします。

投稿者 dog man 時刻: 22:12

ラベル: android, attach, detach, onDraw, onLayout, onMeasure, view, ビュー, レイアウト, 描画

0 件のコメント:

コメントを投稿

コメントを入力...

コメントの記入者: okuda0715tech(

ログアウト

公開 プレビュー

お知らせを受け取る

このブログを検索

検索

自己紹介

dog man

詳細プロフィールを表示

ブログ アーカイブ

人気の投稿