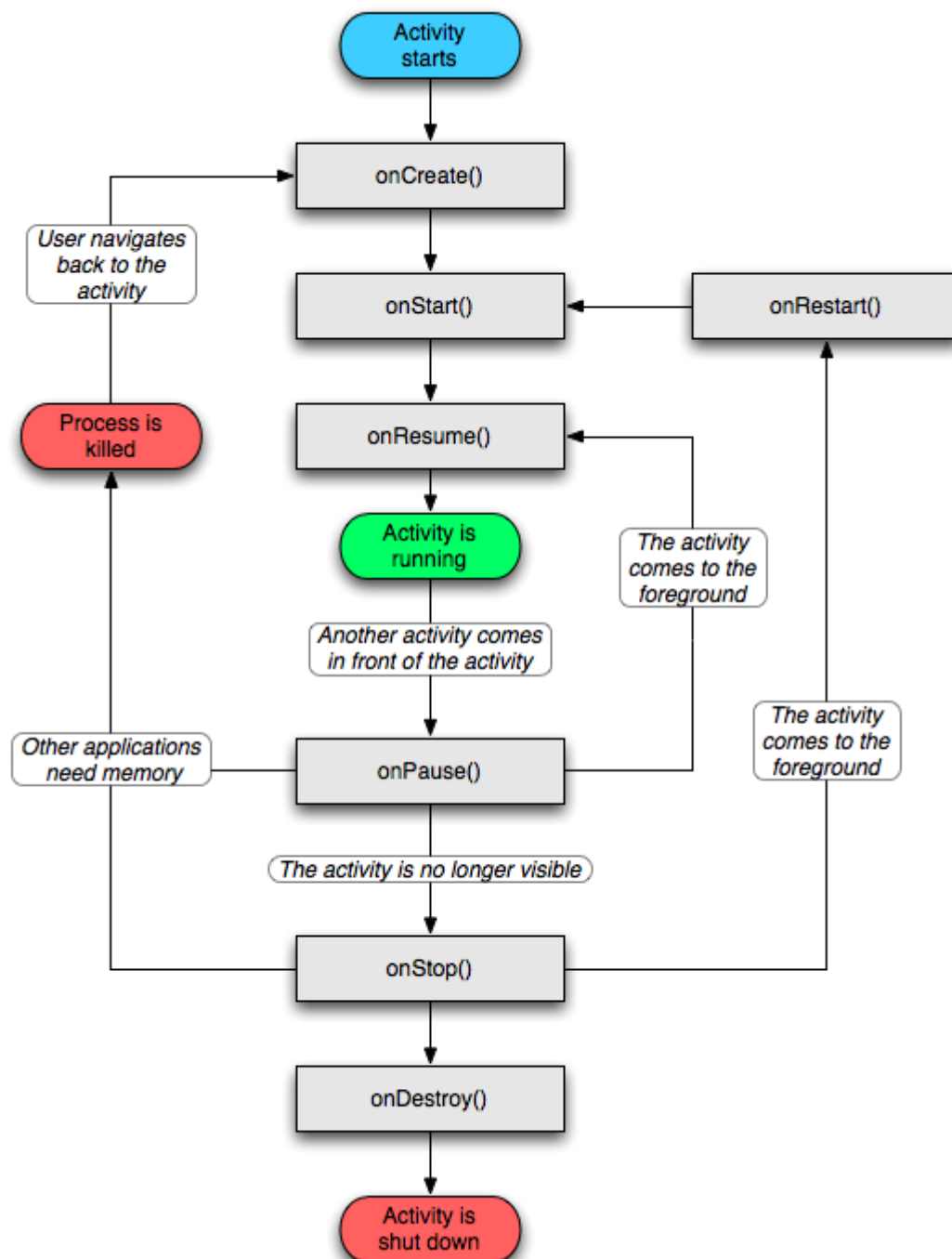


Activity のライフサイクルに関する間違い



onPause の前の「Another Activity comes in front of the activity」という部分は間違い、あるいは間違いで無いとしても非常に紛らわしい。 onPause が呼び出される以前には、

Another Activity オブジェクトは生成されてもいないのである。これは、後述の onPause メソッドの説明からも明らかである。翻訳:「A の onPause から返ってこない限り B は create されないため、ここで長い処理は行ってはならない」。

実際にトレースをおこなってみると、説明の通り、B の onCreate が呼び出されるのは、A の onPause の後である。つまり、以下の順序である。

- A.onPause
- B.onCreate
- B.onStart
- B.onResume
- A.onStop

(A.onPause 以前に B オブジェクト自体が作成されている可能性はあるかもしれないが) B.onCreate が呼び出されるのは A.onPause の後であるため、A.onPause 以前に B が実行可能状態になることは有り得ない。

考えてみればこれは当たり前のとで、A.onPause 以前に B が「実行」されてしまうと、同時に二つの Activity が動作することになってしまう。これでは系統的にもアプリケーションプログラミング的にも複雑になってしまう。

これらをきちんと確認せずに「Activity のライフサイクル」を上の間違った図を元にして説明しているサイトが異常に多い(というよりほとんど)ので注意が必要である。

また、この図には [ConfigurationChange](#) の場合の遷移が記述されていない。すなわち、端末を回転したりした場合には、通常、表示中のアクティビティインスタンスが単純に破棄されて新たなインスタンスが生成される。つまり、

- A.onPause
- A.onStop
- A.onDestroy
- A'.onCreate
- A'.onStart
- A'.onResume

となる。はずだが、なぜか

- A.onPause

- A.onStop
- A.onDestroy
- A'.onCreate
- A'.onStart
- A'.onResume
- A'.onPause
- A'.onStop
- A'.onDestroy
- A".onCreate
- A".onStart
- A".onResume

となってしまう(こちらのバグ?)。

これらのメソッドの意味

なぜこんなにたくさんのメソッドが用意されているか、それらの用途は何なのかが疑問である。

<http://developer.android.com/guide/topics/fundamentals.html>

によると、要するに

- onStart ~ onStop : Activity が可視の状態。
- onResume ~ onPause : Activity がユーザ操作を受けられる状態。

すなわち、onStart~onResume、onPause~onStop の期間は「表示されているけれども、ユーザの操作を受け付けない状態」である。つまりこれは、通常のウインドウシステムにおけるモーダルダイアログが表示されている状態であると考えてよい。

通常のウインドウシステムの場合、これらの期間にウインドウコンポーネントの内容変更を行うと、それらはすぐに表示されるのだが、Android の場合でも 同じなのだろうか？例えばこの期間に [TextView](#) に対して setText を行うと、その表示は変更されるのであろうか？試験方法が不明なので試していない。

ただし、Activity の状態に関わらず、GUI スレッドとは別途に作成した独自のスレッドは動作し続けることができるようである。もちろんこれは Android とは無関係な Java 側の仕様であるので当然だろう。

ただし、onPauseの説明にあるように、onPauseメソッドが呼び出された後は、そのアプリケーションプロセス自体がいつでもkillされる可能性がある。したがって、onStopやonDestroyで「何か」をしようと思うのは意味がない。これらのメソッドは無視されるべきである。onPauseはアプリの終了と同様であると考えなくてはいけない。

つまり、onPause～onStopの期間にActivityの表示を変更するのは意味が無いということになる。何かの表示を行っても、それをユーザが見逃したらこれもまた問題だろう。

また、Activityが作成されてから最初のonResumeに到達するまでのこれらのメソッドはほぼ意味がない。なぜなら、Activityが作成されたにも関わらず、それがonResumeにまで到達しないというのは極めて稀なケースと思われるからだ。つまり、新たなActivityに遷移しようとしてユーザの操作が可能になるまでの間に、電話がかかってきてメモリ不足になるとか、電源ボタンが押されてしまうなどのケースである。

マニュアルにはonStartで[BroadcastReceiver](#)の登録などを行えとあるが、そもそもこの機能自体が特別なアプリケーション向けのものであり、通常はこのようなものは使わないだろう(電話着信やMail着信に反応するアプリがそんなに沢山あってはうざくて仕方がない)し、そもそもユーザとの対話ができない状態なのに、それらの通知を受け取って何の意味があるのだろうか？

また、onRestart無しのonStartはアクティビティの一生の中でただの一度だけしかありえない。したがって、onRestartなど不要であり、onStartかonResumeの引数にフラグでも付ければよい話である。

ふつうのアプリのためのライフサイクルメソッドの省略の仕方

以下のメソッドがあれば十分である。

- **onCreate** : Activityのセットアップを行う。View等の中身は空でよい。データを設定する必要はない。
- **onResume** : 初めてなのかonRestart後であるのかのフラグを渡す。表示すべきデータがあれば保存データを読み出し、Viewに設定する。アニメーションを再開するなど。
- **onPause** : プリファレンス等をセーブする、アニメーションを停止するなど。

サンプルコード

```
package example.android.lifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class ActivityLifecycle extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onResume() {
        super.onResume();
        Toast.makeText(this, "onResume()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onPause() {
```

```

        super.onPause();
        Toast.makeText(this, "onPause()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStop() {
        super.onStop();
        Toast.makeText(this, "onStop()", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "onDestroy()", Toast.LENGTH_SHORT).show();
    }
}

```

■ ライフサイクルの動作を確認してみよう

このアプリは、Activity のイベント通知を受けるメソッドで、以下のように Toast クラスを用いて簡単なメッセージを表示しています。

```

@Override
protected void onPause () {
    super.onPause ();
    Toast.makeText(this, "onPause()", Toast.LENGTH_SHORT).show();
}

```

Android シミュレータで実行して Toast.show()メソッドが呼び出されると、画面に小さなメッセージが表示されます。

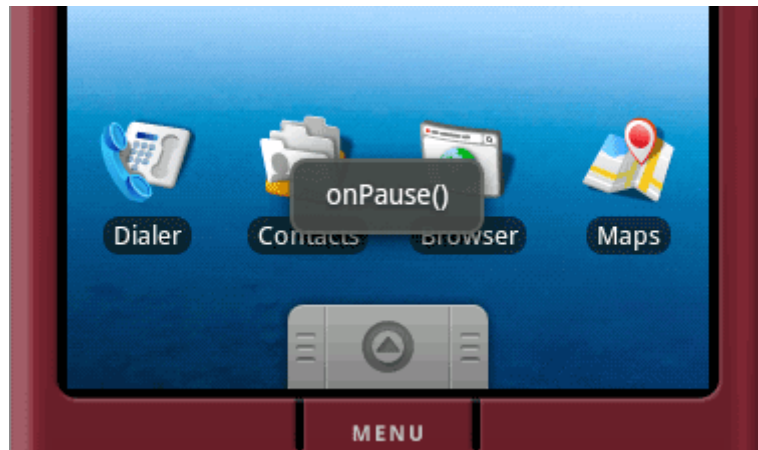


図 1 Toast.show()が呼び出されたところ

Toast は、呼び出し元の Activity が表示されていなくても表示されるので、ユーザーへの通知やデバッグなどに何かと便利です。

さて、アプリを実行すると、状態は「開始」から「実行中」まで遷移し、onCreate()、onStart()、onResume()の Toast が順番に表示されることが確認できたと思います。

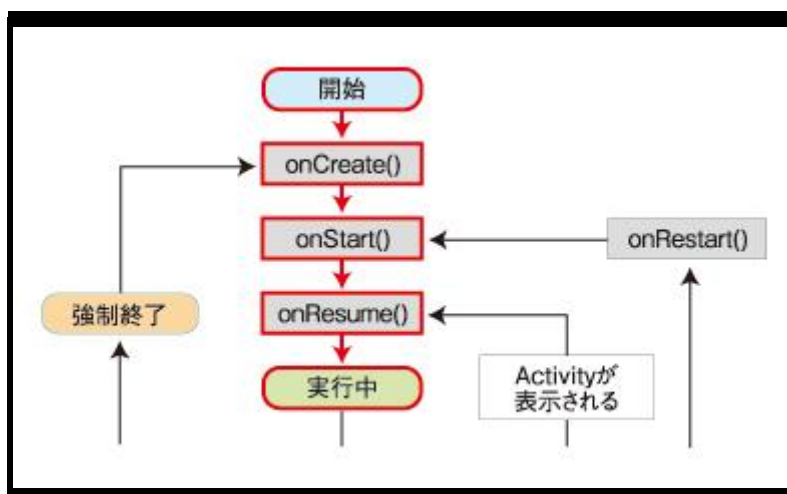



図 2 起動直後の状態遷移

では、実行中の状態で  を押してみてください。このボタンを押すと、電話発信を行う Phone Activity が起動して、元から起動していた Activity が停止状態になり onPause()、onStop()が確認できたと思います。

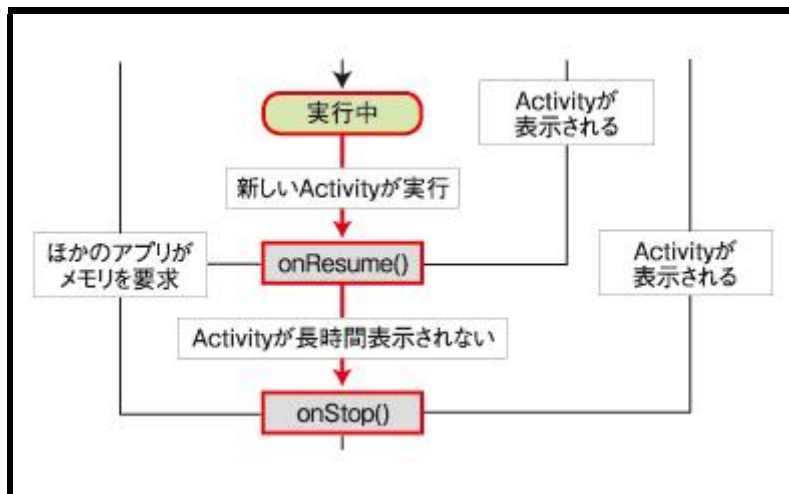



図 3 ほかの Activity を実行した際の状態遷移

onStop()が呼び出された後、Activity は停止状態になります。この停止状態では、ほかのアプリの影響を受けていつでも終了したりメモリが解放されたりする可能性があるため、それらを考慮してプログラミングしなければなりません。

さて、今度は Phone Activity で  を押してみてください。元の Activity に戻って、onRestart()、onStart()、onResume()が確認できたと思います。

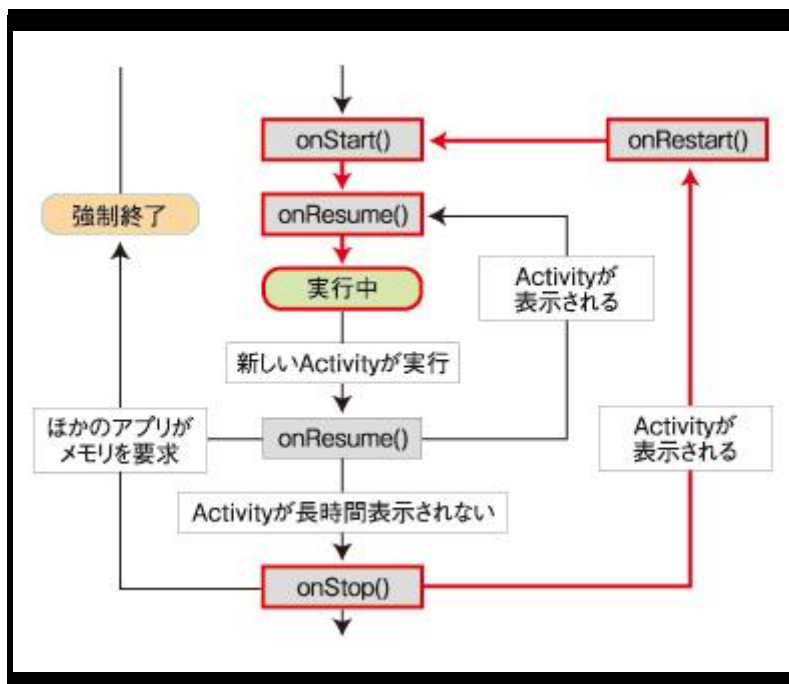



図 4 Activity を再開した際の状態遷移

onStop()からの遷移は、今回は右側の経路を通りましたが、左側の「強制終了」を経由することもあります。どちらを経由するかは、停止中のアプリのメモリ使用量と、そのほか動作しているアプリのメモリ使用量によって変わってきます。

左右どちらの経路をたどったにしても、アプリは再び実行中になりました。ここでまた、を押してみてください。onPause()や onStop()が確認できたと思います。場合によっては onDestroy()も表示されたかもしれません。

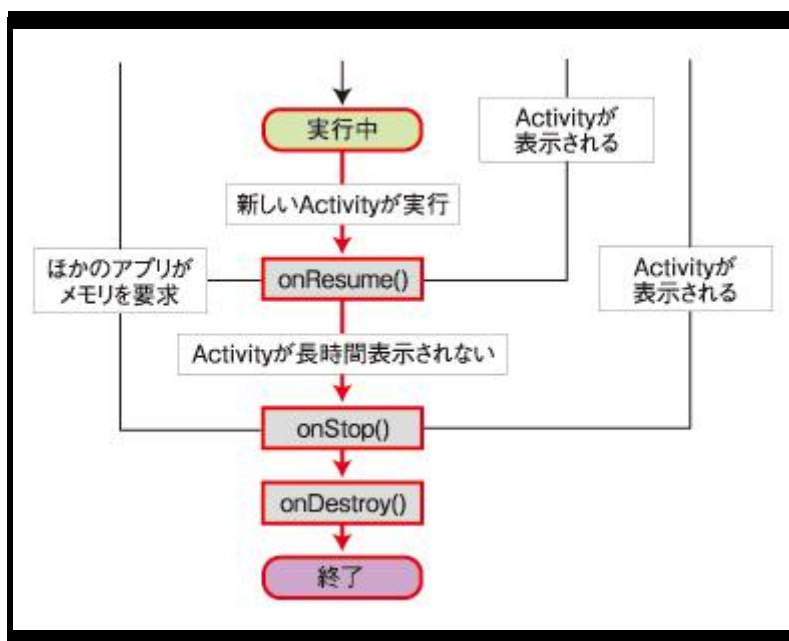



図 5 終了させた場合の状態遷移 (onDestroy()が呼ばれたケース)

Android はアプリを起動して Activity が呼び出され、その Activity から別の Activity が呼び出されると、Activity が**ヒストリースタック**(履歴)に積まれていきます。でヒストリースタックをさかのぼり、ホームスクリーンまで戻るとヒストリースタックは空になります。



Activity を上手にコントロールするには？

Activity の状態が変更されたタイミングでイベントを受け取るコールバックメソッドは、[前述](#)の状態遷移図で紹介しましたが、それら以外にもライフサイクルをコントロールするためのメソッドが用意されているので、一部を取り上げて紹介します。

表 ライフサイクルをコントロールするために使用したいメソッド

メソッド	概要
<code>public void finish()</code>	Activity を終了させる
<code>public boolean isFinishing()</code>	<code>finish()</code> メソッドで終了中かどうかを判定する
<code>public void onLowMemory()</code>	システム全体で空きメモリが少なくなったら呼び出される

■ `finish()`メソッド

`finish()`メソッドは、Activity を終了させるのに使用します。[冒頭](#)で説明したとおり、Activity は Android アプリの画面に当たります。1 つのアプリで複数の画面を持つ場合、A 画面から B 画面を呼び出して、B 画面から A 画面に戻る際に B 画面の `finish()`を呼び出すということもよくあります。

■ `isFinishing()`メソッド

`isFinishing()`は、`finish()`によって状態遷移しているのかそうでないかを判定します。`onPause()`で `isFinishing()`を使用し、必要に応じてアプリの状態を保持するのがよく見られる使い方です。

■ `onLowMemory()`メソッド

`onLowMemory()`はシステムの空きメモリが足りなくなった場合に呼び出されます。このメソッドが呼び出されたら、必要なオブジェクト以外は“破棄”するようにしましょう。このメソッドが呼び出された後に、**ガベージコレクション**が実行されます。