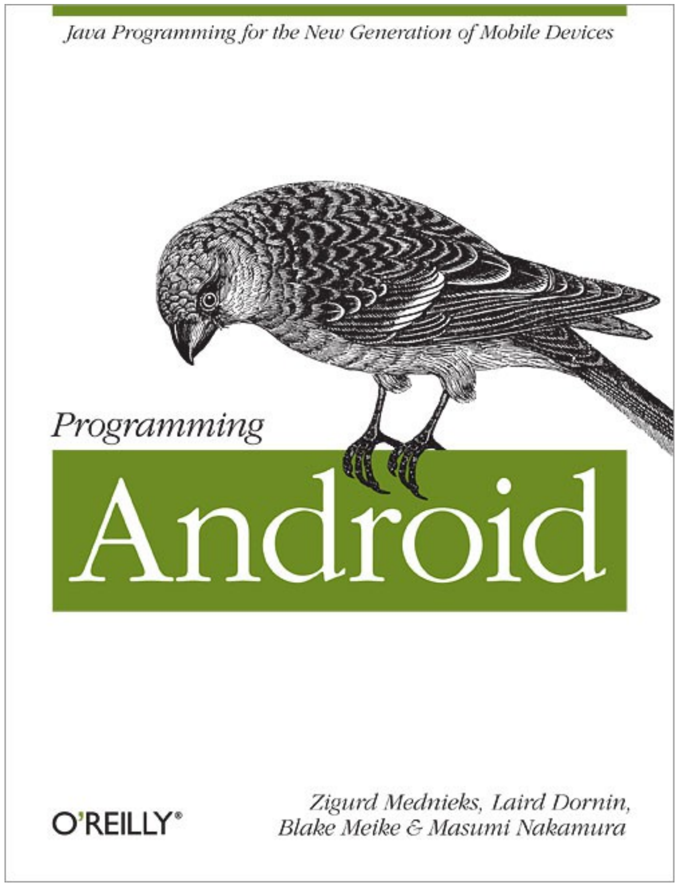
Concurrency In Android

G. Blake Meike

blake.meike@gmail.com

http://portabledroid.wordpress.com/



Java Programming for the New Generation of Mobile Devices

Programming Android

O'REILLY®

Zigurd Mednieks, Laird Dornin, Blake Meike & Masumi Nakamura

Slides and code are available:

git://github.com/bmeike/StrangeLoop.git

marakana

Why is concurrency in Android important?

Until pretty recently, real, concurrent execution happened only on expensive server hardware

- Android already runs on multi-core devices.
- It will get worse: Moore's law is now about number, not speed, of CPUs
- Android devices actually need to do multiple things at the same time: Real Time Computing

Why is concurrency hard?

- It is only hard for us: The real world is very very concurrent. Strict sequence is a strange anomaly perpetrated by Turing, Von Neumann and Minsky, etc.
- Sequential algorithms are like differentiable functions. They are a tiny fraction of all functions, but they are a *really* convenient fraction.
- Concurrency is the relaxation of an un-natural constraint that makes engineering a lot easier
- … but it **is** wicked hard to get it right

Android Puts Concurrency in your Face

- The UI is single threaded but will not tolerate long running, synchronous tasks

- IPC (Binder) calls appear on non-UI threads and you can't use the UI from a non-UI thread
  (a topic for another day...)

Java Concurrency Primitives

- Language level constructs
  - Synchronized
  - Thread/Runnable
- Concurrency library (java.util.concurrent)
  - Executor/Callable/Future
  - Atomics
- All are available in Android
  - Low level Java tools are *really* low level. If you find yourself using them you might want to review your architecture
  - Android Concurrency tools are firmly based on the concurrency library (java.util.concurrent)

Android Concurrency Tools

- AsyncTasks
  - Short running
  - One-shot
  - Used to implement Loaders
- Loopers
  - Long lived
  - Manage task queue
  - Used to implement services (UI thread)
- UI-less Fragments
  - ???
- Renderscript
  - http://marakana.com/s/video_learn_about_renderscript_from_romain_guy_and_chet_haase,381/index.html

Meet AsyncTask

- Type safe template pattern (uses generics)
  - Subclass to use
  - Implement template method: doInBackground
- Designed to make it easy to get short running tasks off the UI thread
  - Most common way to avoid the dreaded ANR
  - Concurrency machinery is hidden

Meet AsyncTask

Async tasks are run on a single, process wide, Executor

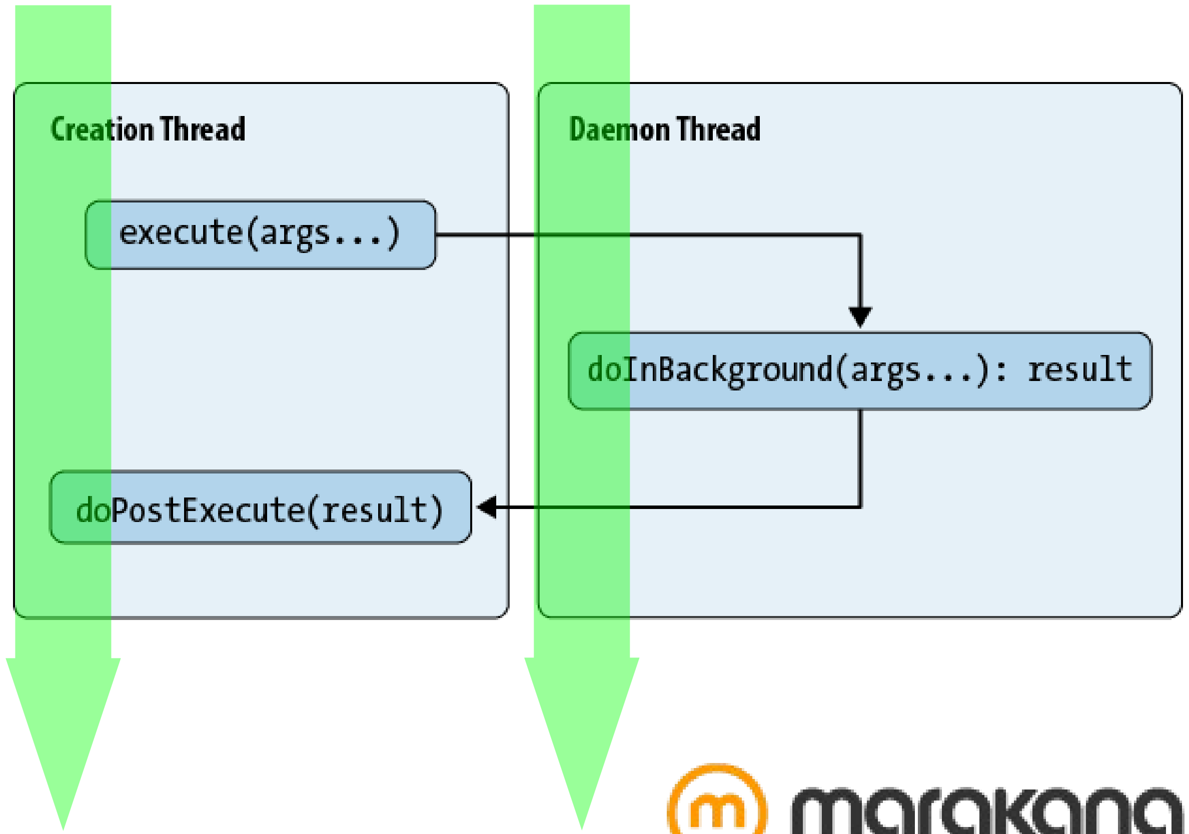
- Between 5 and 128 threads
- API > 12, executes tasks one at at time, in arrival order, to keep time linear. (dequeue)
- executeOnExecutor allows using another Executor

- Android uses AsyncTask all over:
  - Loaders are based on AsyncTask
  - File access
  - Other slow operations

marakana

AsyncTask Lifecycle

- Design: encapsulate a task to be run on a background thread
  - execute
  - onPreExecute
  - doInBackground
  - onPostExecute



marakana

Sample AsyncTask

A very simple AsyncTask, taken directly from Android Webkit code:

```
new AsyncTask<Void, Void, Void>() {
    protected Void doInBackground(Void... none) {
        nativeRemoveSessionCookie();
        signalCookieOperationsComplete();
        return null;
    }
}.execute();
```

So, what could possibly go wrong?

Well...

lots, unfortunately....

This AsyncTask is broken

```
public void initButton(Button button) {
    mCount = 0;
    button.setOnClickListener(
        new View.OnClickListener() {
            @Override public void onClick(View v) {
                new AsyncTask<Void, Void, Void>() {
                    @Override
                    protected Void doInBackground(Void... args) {
                        mCount++;
                        return null;
                    }
                }.execute();
            } });
}
```

This is also broken...

... but for a different reason:

```
public static class BgTask
    extends AsyncTask<String, Void, String>
{
    private final Activity act;
    public BgTask(Activity act) { this.act = act; }

    @Override protected String doInBackground(String... args) {
        // … doesn't matter what goes here...
    }

    @Override protected void onPostExecute(String args) {
        act.onTaskComplete(args)
    }
}
```

So is this!

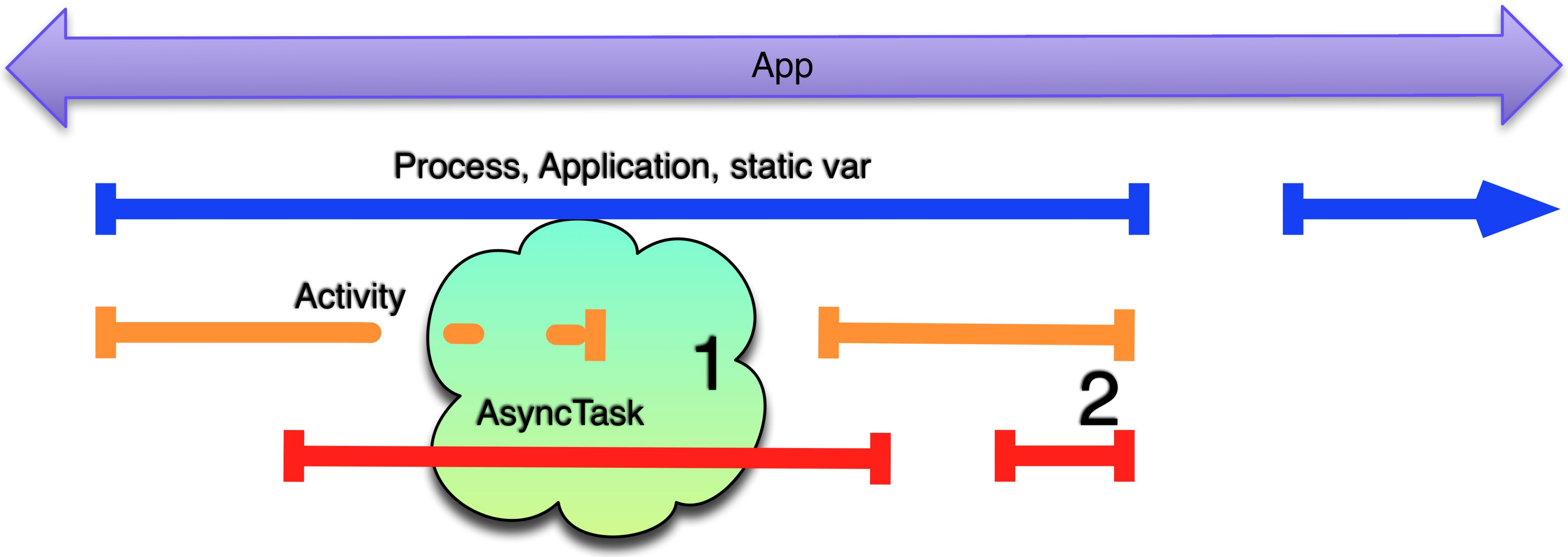
… for the same reason: it's a little bit subtile:

```
public void initButton(Button button) {
    button.setOnClickListener(
        new View.OnClickListener() {
            @Override public void onClick(View v) {
                new AsyncTask<Void, Void, Void>() {
                    @Override
                    protected Void doInBackground(Void... args) {
                        // … doesn't matter what goes here...
                    }
                   @Override
                   protected void onPostExecute(String args) {
                       onTaskComplete(args)
                   }
                }.execute();
            } });
}
```

How to break AsyncTask

- Associated Activity can't be GC-ed until the AsyncTask completes: memory leak
- Associated Activity has been destroyed and is in an inconsistent state: exploding callbacks

Component Lifecycles



1) garbage collection

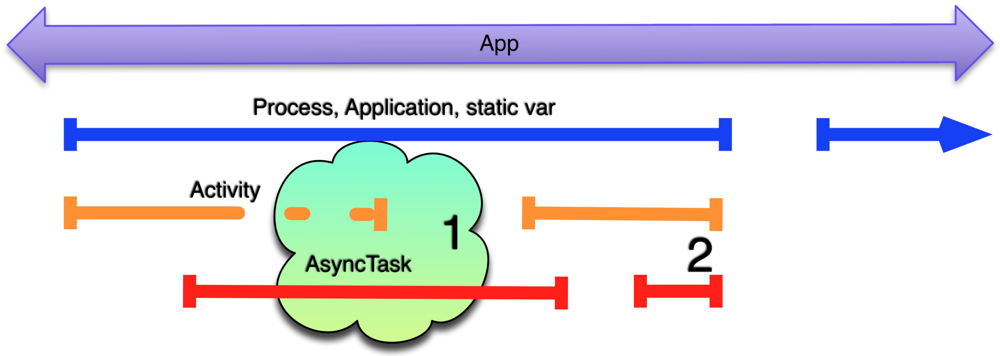
2) process collection

Android Memory Management

There are two kinds of memory management in Android

- Standard Java GC
  - You know and love it.
  - Saw-tooth: memory fills, then GC runs
  - Doesn't affect cross-process memory
- Process Collection
  - Terminates application process
  - Managed by oom_adj

Android Memory Management



Notice:

- It is worth thinking about #1
- There is no point at all in worrying about #2

Possible solutions:

In reverse order of appeal

- Weak Reference
- Persistent
- Cancellable
- Independent

Weak Reference

- Just make all references to the Activity weak
- Currently a “well-known” solution
- The refuge of scoundrels:
  - Allows the process to continue, even when the value will be discarded
  - Doesn't fix references to destroyed processes
- These are the folks who used to lazily initialize their database connections.

Managing a Persistent AsyncTask

```java
public class SafeHeavyweightAsyncTaskActivity
    extends Activity implements HeavywightSafeTask.CompletionHandler
{
    static HeavywightSafeTask task;

    // . . .

    @Override
    public void onTaskComplete(String result) {
        task = null;
        ((TextView) findViewById(R.id.display)).setText(result);
    }

    @Override
    protected void onPause() {
        super.onStop();
        if (null != task) { task.registerCompletionHdlr(null); }
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (null != task) { task.registerCompletionHdlr(this); }
    }
}
```

Persistent AsyncTask I

```java
public final class HeavywightSafeTask extends AsyncTask<String, Void, String> {
    public static interface CompletionHandler { void onTaskComplete(String result); }

    private enum State { EXECUTING, FINISHED, NOTIFIED; }

    private State state = State.EXECUTING;
    private CompletionHandler handler;
    private String result;

    public void registerCompletionHdlr(HeavywightSafeTask.CompletionHandler hdlr) {
        handler = hdlr;
        notifyHdlr();
    }

    @Override
    protected String doInBackground(String... params) {
        // . . .
    }
    @Override
    protected void onPostExecute(String res) {
        result = res;
        state = State.FINISHED;
        notifyHdlr();
    }
    // . . .
```

Persistent AsyncTask II

```
    // . . .

    private void notifyHdlr() {
        if (null == handler) { return; }
        switch (state) {
            case EXECUTING:
                break;
            case FINISHED:
                state = State.NOTIFIED;
                handler.onTaskComplete(result);
                break;
            case NOTIFIED:
                throw new IllegalStateException(
                    "Attempt to register after notification");
        }
    }
}
```

Persistent Task

- Best effort completion
- Hold a reference to the task in static or Application
- Manage references to Activities (onPause, onResume)
- Preserving “single delivery” AsyncTask semantics requires some fancy state management
- Code in GitHub repo
- **The complexity of this solution makes me pretty dubious...**

Managing a Cancellable AsyncTask

```
public class SafeLightweightAsyncTaskActivity
    extends Activity implements LightweightSafeTask.CompletionHandler
{
    LightweightSafeTask task;

    // . . .

    /**
     * @see android.app.Activity#onPause()
     */
    @Override
    protected void onPause() {
        super.onPause();
        if (null != task) {
            task.cancel(true);
            task = null;
        }
    }

    // . . .
```

Cancellable AsyncTask

```java
final class LightweightSafeTask extends AsyncTask<String, Void, String> {

    public static interface CompletionHandler {
        void onTaskComplete(String result);
    }

    private CompletionHandler handler;

    public LightweightSafeTask(CompletionHandler hdlr) { handler = hdlr; }

    @Override
    protected String doInBackground(String... params) {
        // Give up when cancelled!
    }

    @Override
    protected void onCancelled() { handler = null; }

    @Override
    protected void onPostExecute(String ret) { handler.onTaskComplete(ret); }
}
```

Cancellable

- Addresses tasks that can be abandoned
- Implement onCancel
- Call it from onPause

Independent

- What happens when firing the AT is a contract with the UI, and must complete?
- Do the simplest possible proxy to the model
- The inverse of the RESTful ContentProvider cache

Summarizing AsyncTask

- Things that work:
  - References to immutable state
  - References to thread safe state: volatile, synchronized, etc
  - References from things with the same lifecycle: Application, static members, etc
  - Managed references to managed objects

marakana

What's next?

So, personally, I love this kind of stuff.

On the other hand, I've just spent an entire 40 minutes, talking to you about how to do nothing more than accommodate mobile reality.

Are you serious?

Can't we do better than this?

This kind of trivia isn't getting your app into the market

- ...and if you believe me, the bugs are going to get worse.

If you aren't annoyed, you haven't been paying attention!

What's next?

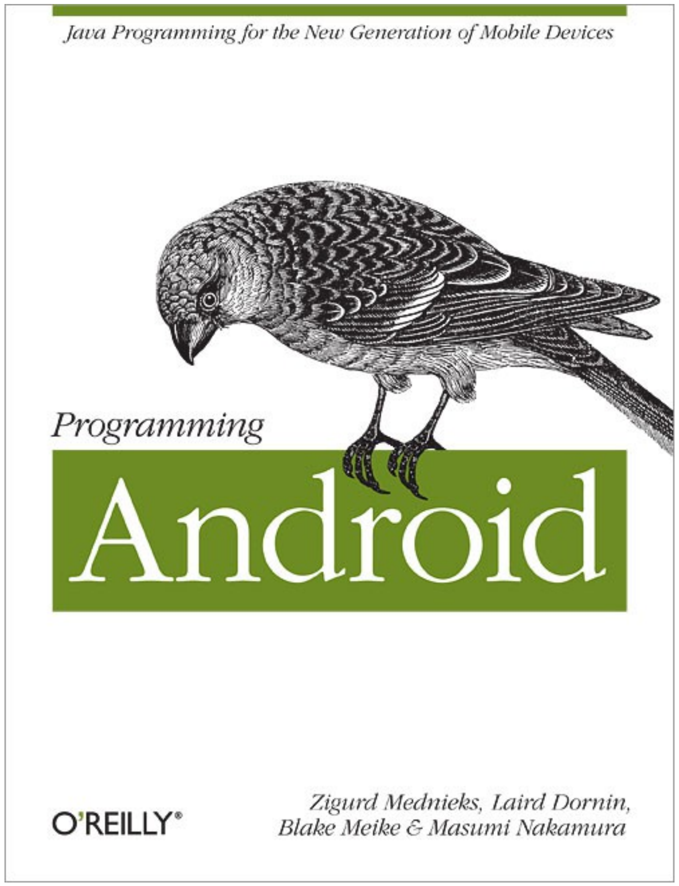
- Immutability?
  - Clojure?
- Actors on Dalvik?
  - Akka on Android?
  - Kilim?
  - Scala?
- Something really new!

Concurrency In Android

G. Blake Meike

blake.meike@gmail.com

http://portabledroid.wordpress.com/


Java Programming for the New Generation of Mobile Devices
Programming
Android
O'REILLY
Zigurd Mednieks, Laird Dornin, Blake Meike & Masumi Nakamura

Slides and code are available:

git://github.com/bmeike/StrangeLoop.git

marakana