Battle-Tested Patterns in Android Concurrency Part 2

Doug Stevenson
Friday, July 31, 2015
11:00 AM

Sample code:
https://github.com/AnDevDoug/concurrency

Recap

- Threads / UI
- AsyncTask
- Started Service
- Looper / Handler / HandlerThread

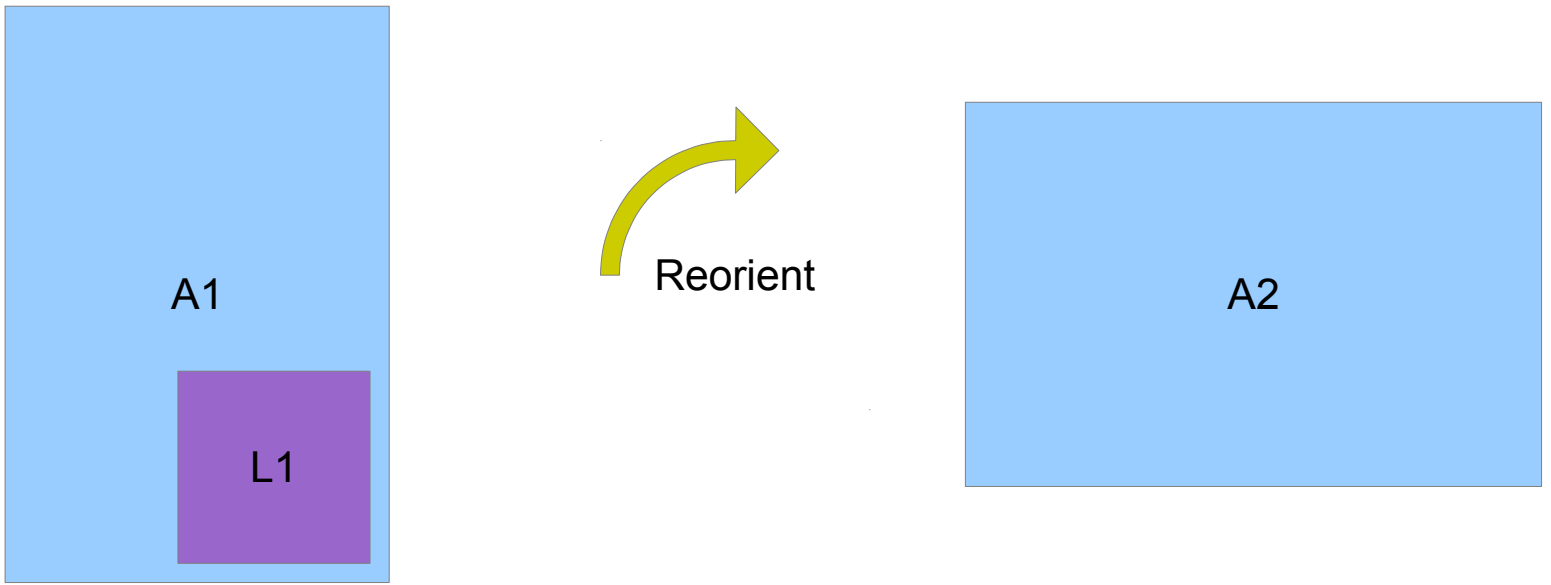
Loaders

Loaders

What they do:

- Put (a single unit of) work on separate thread
- Delivers the result of the work on the main thread
- Continue in-progress work during a configuration change
- Remember work result between activity configuration changes
- Can monitor a source of data for change, notifying the Activity on change (e.g. CursorLoader)
- Signaled when the activity will no longer use its results

Loader Illustrated



Loaders

Use them when:

- You have data to fetch or compute in a single activity that will update the UI
- You have work to do that must survive the Activity lifecycle after a configuration change

For example:

- Load data (from a database, file, network) for display

Loader Notes

Loaders first available in API level 11 (Honeycomb):

- `android.content.Loader`
- `android.app.LoaderManager`
- subclass `android.app.Activity`

Loaders available to API level 4 via Android Compatibility library:

- `android.support.v4.content.Loader`
- `android.support.v4.app.LoaderManager`
- subclass `android.support.v4.app.FragmentActivity`

Loader Usage

Three parts to implementing a Loader:

1. Loader class
    - Subclass of `android.support.v4.content.Loader`
    - Performs background work in another thread
    - Instances managed by LoaderManager
2. LoaderCallbacks class
    - Impl `android.support.v4.app.LoaderManager.LoaderCallbacks`
    - Creates the Loader instance to use
    - Receives the Loader's results, updates the Activity UI

Loader Usage

3. LoaderManager
    - Instance obtained from Activity
        - `getSupportLoaderManager()`
        - `getLoaderManager()`
    - Controls instances of Loaders across Activity config changes

Loader Usage: Loader Class

```
public class YourLoader extends SomeBaseLoader<Result> {

    private final int arg;
    public YourLoader(final Context context, final int arg) {
        super(context);
        this.arg = arg;
    }

    @Override
    protected Result loadInBackground() {
        Result result;
        // use arg to load Result in the background
        return result;
    }

}
```

Tips for Loaders

- May never be non-static inner class in an activity (enforced)
- The Loader implementation decides how to background blocking work

Loader Usage: LoaderCallbacks Class

```
public class YourLoaderCallbacks implements LoaderCallbacks<Result> {

    @Override
    public Loader<Result> onCreateLoader(final int id, final Bundle args) {
        // Create the Loader instance; pass stuff from args into it if necessary
        return new YourLoader(context, args.getInt("key"));
    }

    @Override
    public void onLoadFinished(final Loader<Result> loader, final Result data) {
        // Do something with the loaded result in the UI
    }

    @Override
    public void onLoaderReset(final Loader<Result> loader) {
        // typically empty
    }

}
```

Tips for LoaderCallbacks

- Typically implemented as Activity inner classes
- May contain/use Activity instances without leaking
- But don't pass Activity instances through to the Loader!
- Typically take parameters from the args Bundle
- (but you don't have to pass params that way)

LoaderManager

Manages instances of Loaders between Activity config changes.

For one-time loads, typically done during onCreate():

```
LoaderManager lm = getSupportLoaderManager();
LoaderCallbacks<Result> callbacks = new YourLoaderCallbacks();
Loader<Result> loader = lm.initLoader(
    LOADER_ID,
    (Bundle) null,
    callbacks
);
```

LoaderManager.initLoader()

Two circumstances to remember when you call initLoader():

1. If the Loader with the given id IS NOT already created:
   - The given LoaderCallbacks is asked to create a new one
   - The new loader's onStartLoading() is called
2. If the Loader with the given id IS already created:
   - The given LoaderCallbacks is associated with the existing Loader
   - If the load is already complete, callbacks will be notified next cycle

Other LoaderManager Methods

`Loader<T> getLoader(int id)`

Returns the Loader with the given id or null if not running.

`void destroyLoader(int id)`

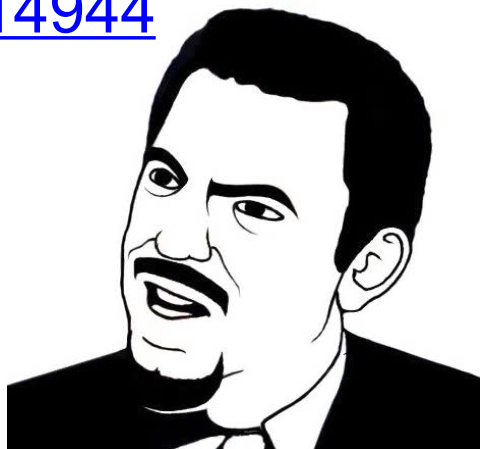
Stops the Loader with the given id.
If already finished work, calls LoaderCallbacks.onLoaderReset.

`Loader<T> restartLoader`
`(int id, Bundle args, LoaderCallbacks<T> data)`

If Loader not already running, works like initLoader.
If Loader already running, it will be destroyed first.

About Android's AsyncTaskLoader

- AsyncTaskLoader is a Loader implementation provided by Android
- Implemented on top of AsyncTask
  - Inherits all of AsyncTask's idiosyncrasies
- Doesn't always work exactly as a Loader should: http://code.google.com/p/android/issues/detail?id=14944



ExecutorServiceLoader: A better loader

- Default operation queues all work on a singleton thread
- Or you can give it an ExecutorService to handle threading
- If you give it an ExecutorService, make it also a global singleton
  - DO NOT create a new ExecutorService in `onCreate()`
- Requires your results to be a `ResultOrException<T, E>`
  - Data container for a generic result type or an Exception subclass
  - Handy because Loaders can't “throw”, but can generate errors
  - Must check if result or exception exists before using either

Basic Loader Demo

See ActivityBasicLoader.java

Loader as a Non-static Inner Class

Loader helps defend against accidental Activity leaks:

- All Loader classes are required NOT to be a non-static inner class.
- Non-static inner class loaders will make your app will crash:

```
java.lang.IllegalArgumentException: Object returned from onCreateLoader
must not be a non-static inner member class
```

- Nothing stopping you from injecting an Activity into a custom loader.

Non-Static Loader Demo

See ActivityInvalidNonStaticLoader.java

A Tricky Situation with a Loader

1. You have a Button that:
   - Kicks off a Loader (NOT in onCreate())
   - Updates UI to disable the button and show a wait spinner
2. Configuration change → new Activity
3. New Activity needs to reattach the Loader and disable the button and show the spinner

Problems:

- You can't blindly call initLoader() in onCreate()
- getLoader() won't tell you if the loader is in progress or finished

Stateful Loader Part 1

Create a Loader subclass with a method getState() that returns an enum for load state (e.g. Loading, Loaded).

```
public class StatefulLoader extends BaseLoader<Result> {

    private volatile State state;
    public static enum State { Loading, Loaded; }
    public State getState() { return state; }

    protected Result onLoadInBackground() {

        state = State.Loading;
        // Do your loading here
        state = State.Loaded;

        return result;

    }
}
```

Stateful Loader Part 2

Then in onCreate():

```
initViews();

LoaderManager lm = getSupportLoaderManager();
Loader<Result> loader = lm.getLoader(LOADER_ID);
StatefulLoader statefulLoader = (StatefulLoader) loader;

if (statefulLoader != null) {

    lm.initLoader(LOADER_ID, null, new YourLoaderCallbacks());

    switch (statefulLoader.getState()) {
    case Loading:
        updateUiLoading();
        break;
    case Loaded:
        break;
    }

}
```

Stateful Loader Demo

See ActivityStatefulLoader.java

Another Trick for Saving Loader State

How to keep track of multiple potential loaders?

1. Remember all Loader ids that have been init'd
2. In onSaveInstanceState(), save all init'd loader ids in the Bundle
3. In onCreate():
    a) Get list of saved loader ids from the Bundle arg
    b) Check their state, update UI
    c) Init each each loader id again

Tracking Loader Work Progress

If you have a Loader that should track incremental progress:

- Start with Stateful Loader pattern
- Use LocalBroadcastManager as a data exchange mechanism
- In the Loader, broadcast progress updates
- Implement a BroadcastReceiver to handle progress updates
- Register the BroadcastReceiver in onCreate() / Unregister in onDestroy()

Tracking Loader Demo

See ActivityProgressLoader.java

CursorLoader: Loading from ContentProvider

To use Android's CursorLoader, you need a Uri for a ContentProvider:

- From an Android system component (Calendar, Contacts, Media)
- From another app
- One you created for yourself

http://developer.android.com/guide/topics/providers/content-providers.html

CursorLoader Usage

1. Create a LoaderCallbacks class that implements `LoaderCallbacks<Cursor>`
2. In `onCreateLoader()`, create and return a CursorLoader with the ContentProvider query
3. In `onLoadFinished()`, make use of the Cursor

CursorLoader Callbacks Example

```
public class YourLoaderCallbacks implements LoaderCallbacks<Cursor> {

    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        return new CursorLoader(activity, content_uri, ...);
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        listView.setAdapter(new YourCursorAdapter(activity, data, 0));
    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
    }

}
```

CursorLoader Demo

See ActivityMusicCursorLoader.java

Loaders and Asynchronous APIs

Using a fully asynchronous API?

(methods return immediately work is on another thread, calls a listener on completion)

For example:

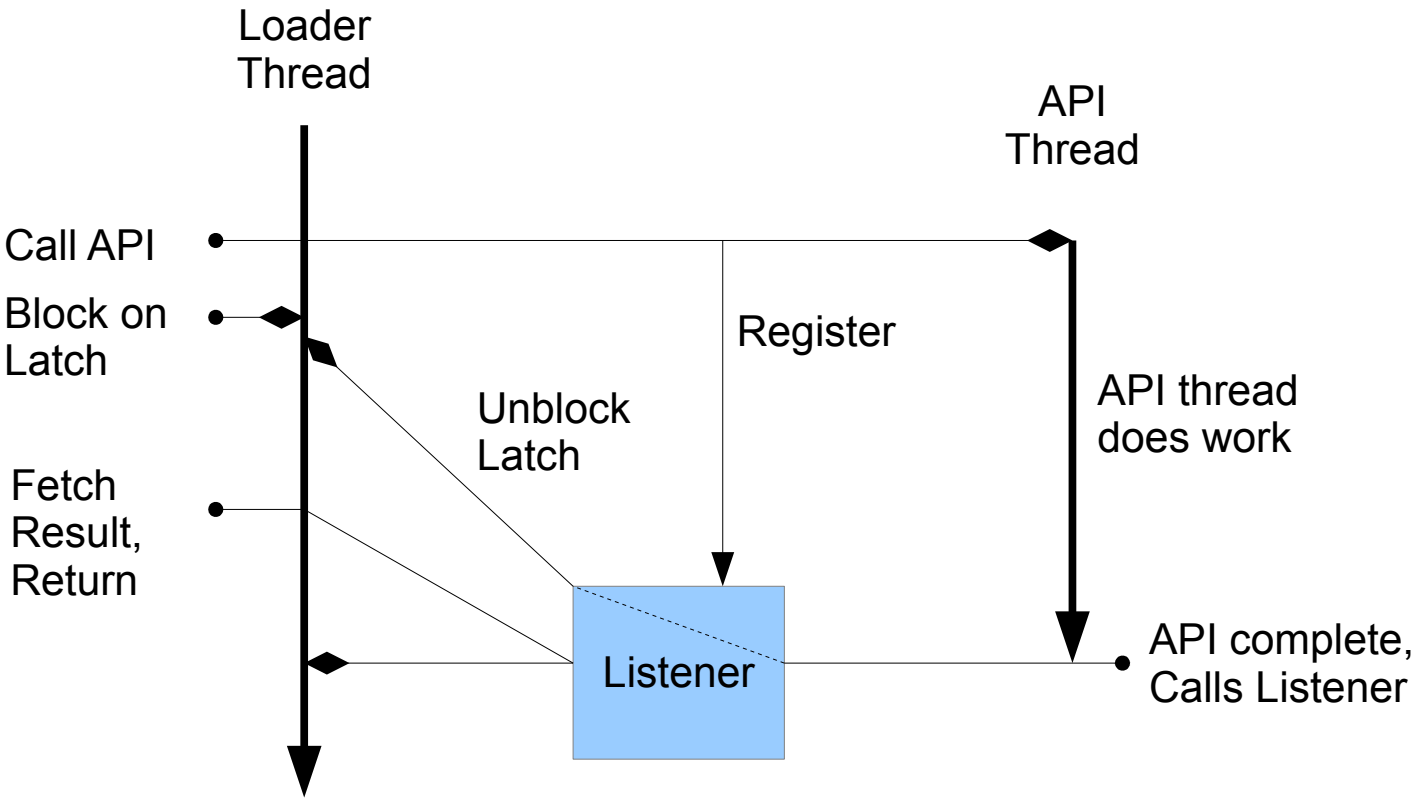
```
SocialNetworkApi api = …

api.loadProfile(new LoadProfileCallback(
    public void onResult() {
        // update UI with result
    }
));
```

Loaders and Asynchronous APIs

1. Call the API in onLoadInBackground()
2. Use a CountDownLatch to block the Loader thread and wait for a result
3. Have the API listener signal the Latch; store the API result
4. After the Latch unblocks the loader thread, fetch the result and return it

Loaders and Asynchronous APIs



Async API Loader Demo

See ActivityAsyncApiLoader.java

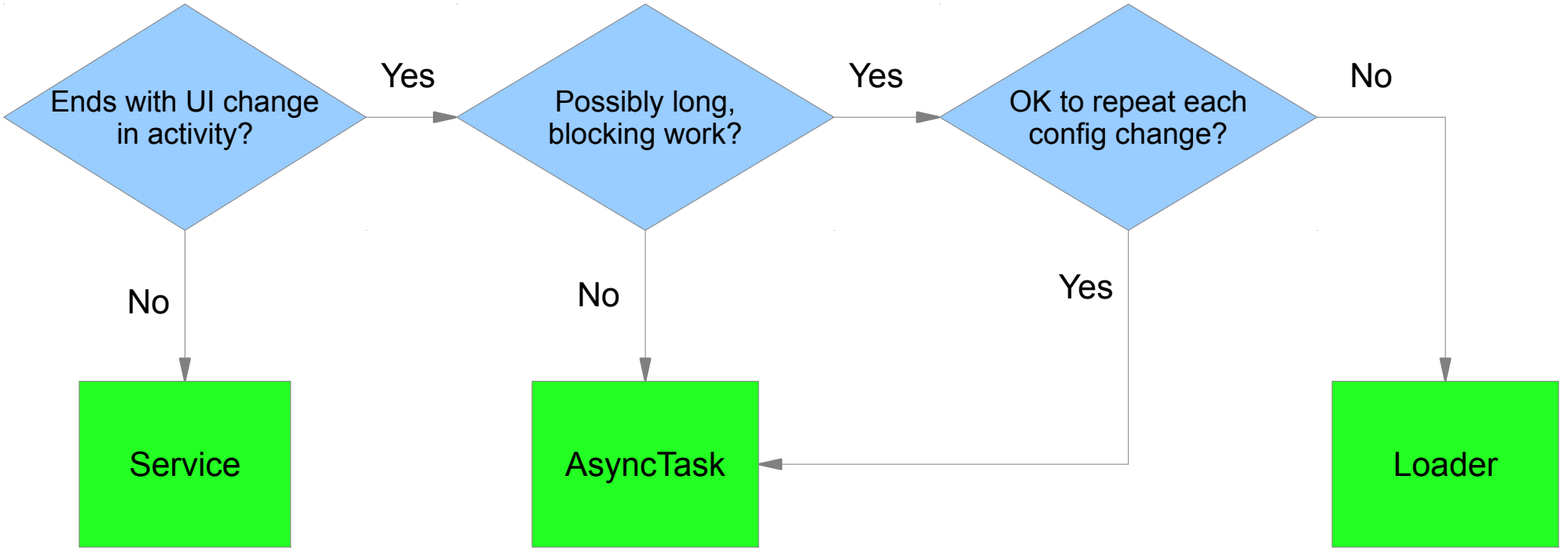
When Not to Use a Loader

- Lots of little downloads (e.g. thumbnails)
  - Consider Volley, Picasso or another framework
- The work needs to continue after the activity is done

Loader Summary

- Loader is a useful and underutilized tool for Android development.
- Some boilerplate overhead in coding
- Addresses the most common problems with background tasks in Activities

What Tool to Use?



Optimizing Threading Behavior

Optimizing Intermittent Network I/O

- e.g. High throughput remote API calls with small payloads
- Limit to two or three threads to prevent saturating a slow connection
- Maybe increase threads if connection speed is high

Optimizing Sustained Network I/O

- e.g. Downloading large files and images
- Limit to just one thread to prevent saturating a slow connection

Optimizing File I/O

- e.g. Simple database queries that can touch many rows
- e.g. Access to external storage
- Limit to just one thread (per storage device) to prevent I/O thrashing

Optimizing Heavy CPU Work

- e.g. Decoding many/large bitmaps (or any media)
- e.g. Performing complex database operations (could be I/O intense as well)
- Limit to number of CPU cores minus one
  - `Runtime.getRuntime().availableProcessors();`

Strict Mode

Force your app to crash/alert when behaving badly:

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
    .detectAll()   // disk read/write, network
    .penaltyLog()
    .penaltyDeath()
    .build());

StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
    .detectAll()   // leaked activities, sqlite cursors, closables
    .penaltyLog()
    .penaltyDeath()
    .build());
```

Thread Priorities

- Wisely use `android.os.Process.setThreadPriority(prio)`
  - `android.os.Process.THREAD_PRIORITY_BACKGROUND`
  - `android.os.Process.THREAD_PRIORITY_AUDIO`
  - (Avoid `java.lang.Thread.setPriority()`)

Other concurrency tips

- Android has all the same thread tools and behavior as Java 5.
  - Semaphore, BlockingQueue
  - ConcurrentHashMap, CopyOnWriteArrayList, skip lists
- Devices may not power up all their CPU's cores immediately
- Avoid polling loops at any cost
- Consider Renderscript Computation (API 11+) to offload heavy math to the GPU
  - http://developer.android.com/guide/topics/renderscript/compute.html

Other concurrency tips

- Consider RxJava for heavy duty concurrency
- Upsides
  - Complex pipelines and transformations
  - Probably better overall concurrency
- Downsides
  - Big jar (by mobile perspective)
  - Complex
  - Hard to debug

Feedback?
eventmobi.com/adcboston

Sample code:
https://github.com/AnDevDoug/concurrency