Android Developer Big Bag of Tricks

Doug Stevenson
Thursday July 30 @ 8:00 am

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

My Office Hours: Thursday 5:30 – 6:00pm

Battle-Tested Patterns in Android Concurrency
Friday 8:30
In two thrilling sessions!

Session Topics

- App initialization
- Logging
- Configuration
- Dev/Test Utility

What is your Application's `Application`?

android.app.Application

- Singleton
- Is a subclass of `android.content.Context`
  (like Activity, Service, BroadcastReceiver)
- Don’t assume to use it anywhere a Context is accepted
- Custom subclass declared in manifest

Application First

- `onCreate()` is called on the main thread before any other code
  - Except for every `ContentProvider.onCreate()`!
- Never block here

A Custom Application

```
package mypackage;

public class MyApplication extends Application {
    public void onCreate() {
    }
}
```

A Custom Application

```
<application
    android:name="mypackage.MyApplication"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
```

Things to do in Application.onCreate()

- Init SDKs and libraries
- Inject Application instance elsewhere
- Register dynamic broadcast receivers
- Arrange for background work

What about Activity.onCreate()?

```
<activity android:name="SomeActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>

<service android:name="SomeService">
    <intent-filter>
        <action android:name="SOME_ACTION" />
    </intent-filter>
</service>

<receiver android:name="SomeReceiver">
    <intent-filter>
        <action android:name="android.intent.action.SCREEN_ON" />
    </intent-filter>
</receiver>
```

Launch Execution Order

1. Every `ContentProvider.onCreate()`
2. `Application.onCreate()`
3. Invoked component `onCreate()`
   - Activity, BroadcastReceiver, Service

(Don’t use Activity.onCreate for at-launch work.)

Getting the Application

- Requires an existing Context
  `context.getApplication()` / `context.getApplicationContext()`
- Is `instanceof` your Application subclass
- May hold strong references to it indefinitely
  (Application instances can not be leaked)

(Better) Logging

Android Logging API

```
import android.util.Log;

Log.d("TAG", "Message", Throwable);
Log.i("TAG", "Message", Throwable);
Log.v("TAG", "Message", Throwable);
Log.w("TAG", "Message", Throwable);
Log.e("TAG", "Message", Throwable);
Log.wtf("TAG", "Message", Throwable);
```

Problems with Log.*

- Causes local unit tests to fail:

```
java.lang.RuntimeException: Method d in android.util.Log not mocked.
See https://sites.google.com/a/android.com/tools/tech-docs/unit-
testing-support for details.
    at android.util.Log.d(Log.java)
```

- Won’t work in java libraries to be shared outside Android

Bridge Java Logging API

- Implement a Handler that routes Java log messages to logcat
- Register the handler at app launch
- Java Logging is configurable

Init Logging during Application.onCreate()

```
private void initLogging() {
    String pkg = getClass().getPackage().getName();
    AndroidLogHandler alh = new AndroidLogHandler(pkg);
    Logger logger = Logger.getLogger(pkg);
    logger.addHandler(alh);
    logger.setUseParentHandlers(false);
    logger.setLevel(Level.FINEST);
    logger.info("Logging initialized with default level " +
                logger.getLevel());
}
```

Log (not) All the Things

```
import java.util.logging.Level;
import java.util.logging.Logger;

public class MySuperbActivity extends Activity {
    private static final Logger LOGGER =
        Logging.getLogger(MySuperbActivity.class.getName());

    private void aloha(String name) {
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.fine("Initiating greet sequence for " + name);
        }
    }
}
```

Other Logging Options

- log4j/slf4j
- logback/slf4j
- But do you want even more dependencies?

(Better) Config Changes

Config Change Refresher

Activities can be destroyed and recreated by:

- Display/Orientation
- Telephony changes
- Memory pressure
- See doc for `<activity android:configChanges>`

Saving Activity State

- `onSaveInstanceState(Bundle)`
- `onCreate(Bundle)` or `onRestoreInstanceState(Bundle)`
- Bundle only deals with primitive types and arrays
- Can be a lot of manual code for all activities

The Painful Way

```
private String fooString;
private int counterInt;
private String[] someThings;
. . .

public void onSaveInstanceState(Bundle b) {
    b.putString("fooString", fooString);
    b.putInt("counterInt", counterInt);
    b.putStringArray("someThings", someThings);
    // ad nauseam...
}
```

The Painful Way

```
protected void onCreate(Bundle b) {
    fooString = b.getString("fooString");
    counterInt = b.getInt("counterInt");
    someThings = b.putStringArray("someThings");
    // ad nauseam...
}
```

An Easy Way - Declare

```
static class State implements Serializable {
    private String fooString;
    private int counterInt;
    private String[] things;
}

private State state;
```

An Easy Way - Save

```
protected void onSaveInstanceState(Bundle b) {
    BundleSerializer<State> serializer =
        new BundleSerializer<State>();
    serializer.serialize(state, b);
}
```

An Easy Way - Restore

```
protected void onCreate(Bundle b) {
    if (b != null) {
        BundleSerializer<State> serializer =
            new BundleSerializer<State>();
        state = serializer.deserialize(b);
    }
    else {
        state = new State();
    }
}
```

BundleSerializer Implementation

- Given: Java Serialization (ObjectInputStream/ObjectOutputStream)
- JSON Serialization (GSON)
- Write something specialized

App Configuration

Where do you stash your constants?

Why think about this?

- White label apps
- Same app, different stores
- Free / Paid
- Dev / Beta / Release
- Customer overrides

What could change?

- API Keys
- Web service host/port
- Log level
- Dev / Debug tool availability

Your options:

- Hard code constants in Java
- Code gen?
- Put constants in Android resources

Configuring with Resources

- Primary/defaults: `res/values/config.xml`
- Dev overrides: `res/values-v1/config.xml`
- Tip: ignore `res/values-v1` in SCM to prevent dev env checkins

Configuring with Resources

- Gradle build resource injection:
  `resValue "string", "name", "value"`

- Use under gradle config hierarchy:
  - android.defaultConfig
  - android.buildTypes.*
  - android.signingConfigs.*
  - android.productFlavors.*

Loading Config Resources

- Application.onCreate() once again
- Load res into object model
- Maybe more config changes at runtime?

Dev/Test/QA Tooling

Dev/Test/QA Tooling

- Quickly exercise app features and edge cases; faster dev cycles
- Simulate complex use cases
- Change app configuration on the fly
- Must be missing from prod builds

“dev” Product Flavor - Gradle

```
productFlavors {
    // prod flavor to be shipped to market
    prod {
        versionName '1.0'
    }

    // dev flavor for internal use only
    dev {
        versionName '1.0-dev'
    }
}
```

“dev” Product Flavor - source

```
APP_ROOT
`- src
   `- dev
      |- java
      |  `- com.company
      |     `- DevLaunchActivity.java
      |- res
      |  `- layout
      |     `- activity_dev.xml
      `- AndroidManifest.xml
```

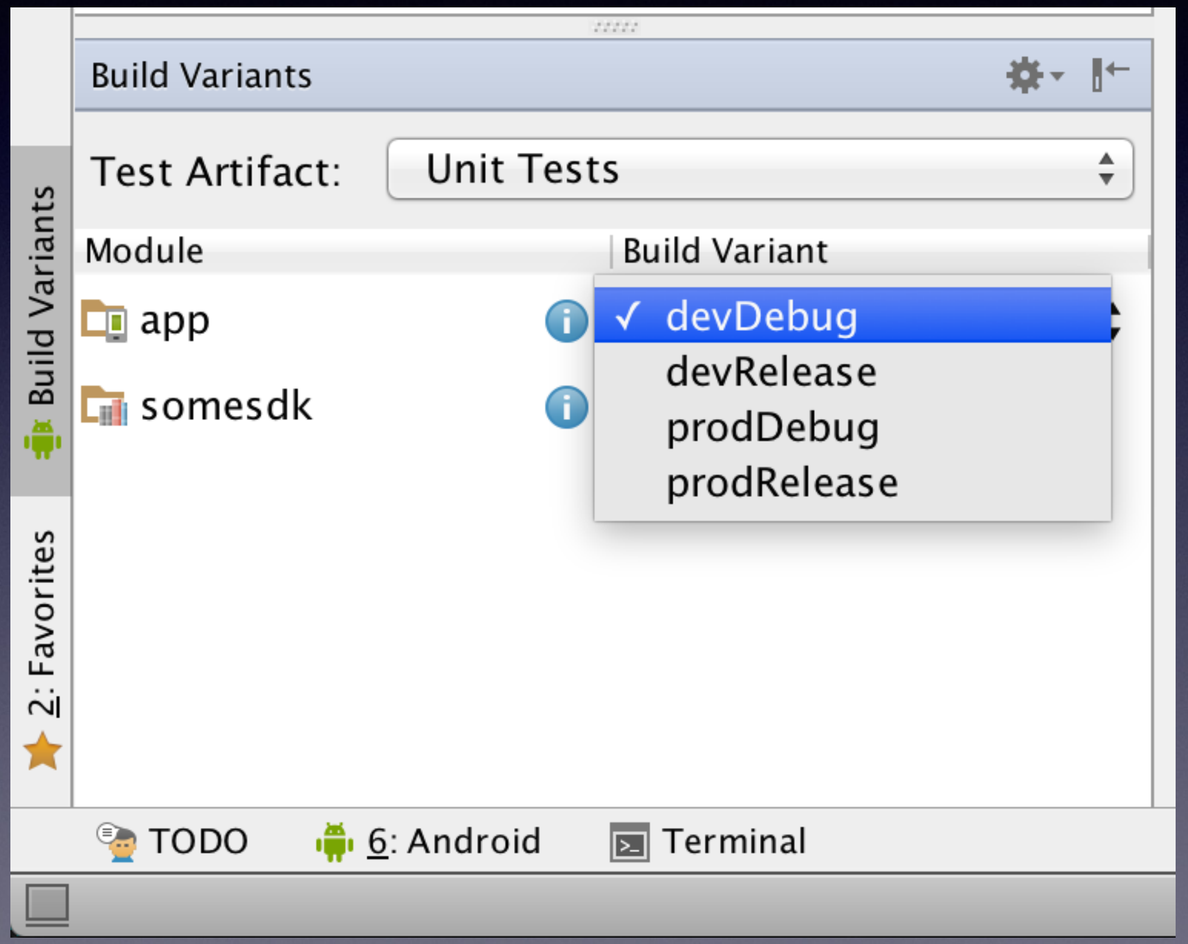
“dev” Product Flavor - manifest

```
<manifest>
    <application>
        <activity
            android:name=".DevLaunchActivity"
            android:label="Dev Launcher" >
            <intent-filter>
                <action android:name="MAIN" />
                <category android:name="LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

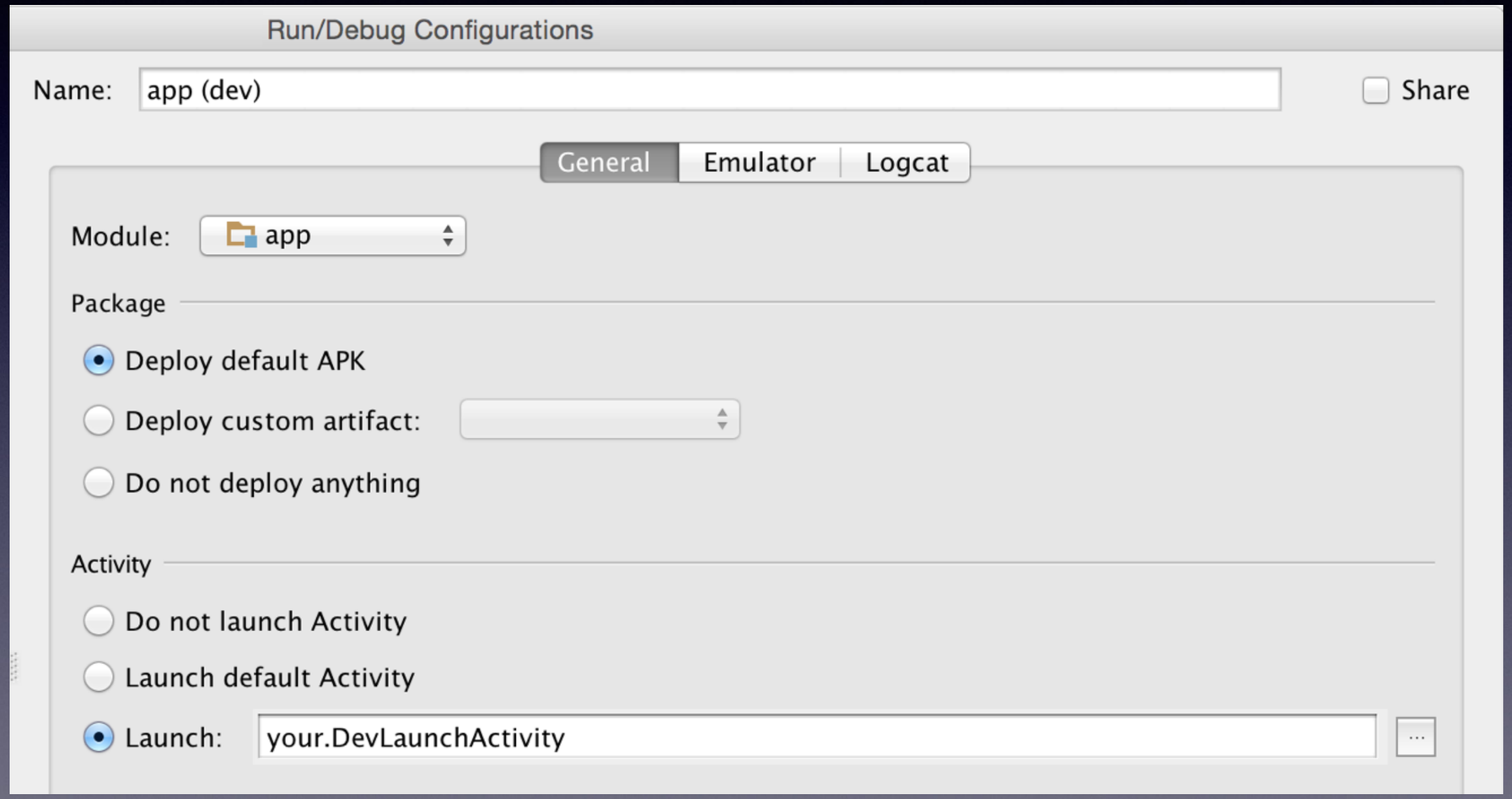
Manifest Merger Magic

- It is magic.

Flavor/type Selector



Custom Run Config



Feedback?
eventmobi.com/adcboston

My Office Hours
Thursday 5:30 – 6:00pm

Battle-Tested Patterns in Android Concurrency
Friday 8:30am
In two thrilling sessions!