Scheduling Periodic Tasks

What is a periodic task?

Yes

1 minute < period < 1 day

A single code path executed multiple times

Usually asynchronous WRT the UI

No

Loading avatars for a ListView (Volley)

Populating a ListView from a database (Loader)

Receiving a system status update (Receiver)

Wish list

- Thread-safe
- Aware of component life-cycles
- Smart use of process priority
- Thrifty with power

Thread safe

Java thread safety is hard and boring:

It is a perfect job for a computer.

Component lifecycle aware

Android components are managed by Android.

An independently executing task cannot count on any currently existing components being present.

Schedule should be persistent across processes and reboots

Smart process scheduling

A visible Activity has high priority

A “started” Service has medium priority

An invisible Activity has low priority

Thrifty with power

A task scheduled when the device is otherwise asleep had better seize a wakelock!

Better to wake the device up once, than multiple times

Better to schedule all network use at once

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty w/power
TimerTask				
Looper/Handler				
IntentService				
AlarmService				
SyncAdapter				
JobService				

TimerTask

Use a Java TimerTask to schedule periodic execution.

TimerTask

Dead, dead, dead, dead.

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty power
TimerTask				
Looper/Handler				
Intent/Alarm Service				
SyncAdapter				
JobService				

Looper/Handler

Use `sendMessageDelayed` to post a message for execution at a future time.

```
private volatile boolean stop;
private HandlerThread looper;
private Handler handler;

public void periodicTask() {
  stop = false;
  looper = new HandlerThread(TAG);
  looper.start();
  handler = new Handler(looper.getLooper()) {
    @Override
    public void handleMessage(Message msg) {
      if (stop) { return; }
      switch (msg.what) {
        case TASK:
          runTaskOnBgThread();
          handler.sendEmptyMessageDelayed(TASK, INTERVAL_MS);
          break;
      }
    };
  handler.sendEmptyMessage(TASK); // start here!
}
```

Looper/Handler: hot

- Very very lightweight: Messages are pooled
- Exact scheduling down to the millisecond
  … deviation on UI thread is up to 100ms

Looper/Handler: not

- A bit bulky
- No app-wide thread policy
- As thread-safe as you are.
- Utterly ephemeral

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty Power
TimerTask				
Looper/Handler				
Intent/Alarm Service				
SyncAdapter				
JobService				

Roll your own

Build a bindable service with a ScheduledThreadPoolExecutor.

Start the service when there are tasks in the queue.

Roll your own: details

Out of scope. You do what you want.

Roll your own: hot

- Can work quite well
- You have complete control over the features

Roll your own: not

- Limited access to power/networking information
- You are pretty much on your own.
- You own all the bugs
- You get nothing free

Alarm Manager/Intent Service

Use the Alarm Manager to post Intents, periodically, to an IntentService

```
public static void startPeriodicTask(Context ctxt) {
   ((AlarmManager) ctxt.getSystemService(Context.ALARM_SERVICE))
    .setInexactRepeating(
      AlarmManager.RTC,
      System.currentTimeMillis() + 100,
      ctxt.getResources().getInteger(R.integer.poll_interval),
      getTaskIntent(ctxt));
}

public static void stopPeriodicTask(Context ctxt) {
  ((AlarmManager) ctxt.getSystemService(Context.ALARM_SERVICE))
    .cancel(getTaskIntent(ctxt));
}

private static PendingIntent getTaskIntent(Context ctxt) {
  Intent i = new Intent(ctxt, TaskService.class);
  i.putExtra(PARAM_OP, OP_TASK_1);
  return PendingIntent.getService(
    ctxt,
    TASK_ID,
    i,
    PendingIntent.FLAG_UPDATE_CURRENT);
}
```

AlarmManager: details

- Will wake the device from dead sleep
- Choose RTC or Elapsed Scheduling
- Clusters tasks with inexact calls on API > 19
- Jitter! Do not schedule all of your apps to phone home at exactly 2am on Sundays!
- Persistent across restarts but not reboots.

```
@Override
protected void onHandleIntent(Intent intent) {
  int op = intent.getIntExtra(PARAM_OP, 0);
  switch(op) {
    case TASK_1:
      helper.doTask1();
      break;

    case TASK_2:
      helper.doTask2(
        intent.getStringExtra(PARAM_ARG));
      break;

    default:
      Log.d(TAG, "Unrecognized op: " + op);
  }
}
```

IntentService: details

- Very nice isolation
- Tasks executed on a single Looper thread
  (in-order: frequently exactly what you want)
- Service “started” when doing work

A Word about Wakelocks

If your periodic task is the only reason the device is powered up, it will need to hold a wakelock

The Alarm manager holds the wakelock long enough to deliver the intent

…but not long enough to start the service!

- WakefulBroadcastReceiver
- Mark Murphy's WakefulIntentService

Alarm/IntentService: hot

- Isolation is good for thread safety
- Isolation is good for component safety
- Inexact scheduling is pretty power-smart
- Some support for power management

Alarm/IntentService: not

- Isolated!
- Wakelock management
- Need rescheduling after install or reboot

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty Power
TimerTask				
Looper/Handler				
Intent/Alarm Service				
SyncAdapter				
JobService				

Take a step back

The preceding frameworks are all very low level:

We've been talking about the implementation, not the purpose.

Let's take it up a level…

Sync Adapter

Connect a local dataset, named by a `content://` URL, with a credentialed remote account.

From: https://github.com/bmeike/MiniSync
A minimal SyncAdapter

```
<sync-adapter
xmlns:android
  ="http://schemas.android.com/apk/res/android"
android:accountType="@string/account_type"
android:contentAuthority
  =“io.realm.android.minisync”
android:isAlwaysSyncable="true" />
```

```
<service
  android:name=".SyncService"
  android:exported="false">
  <intent-filter>
    <action android:name=
      "android.accounts.AccountAuthenticator"/>
    <action android:name="android.content.SyncAdapter"/>
  </intent-filter>

  <meta-data
    android:name="android.accounts.AccountAuthenticator"
    android:resource="@xml/account"/>

  <meta-data
    android:name="android.content.SyncAdapter"
    android:resource="@xml/sync"/>
</service>
```

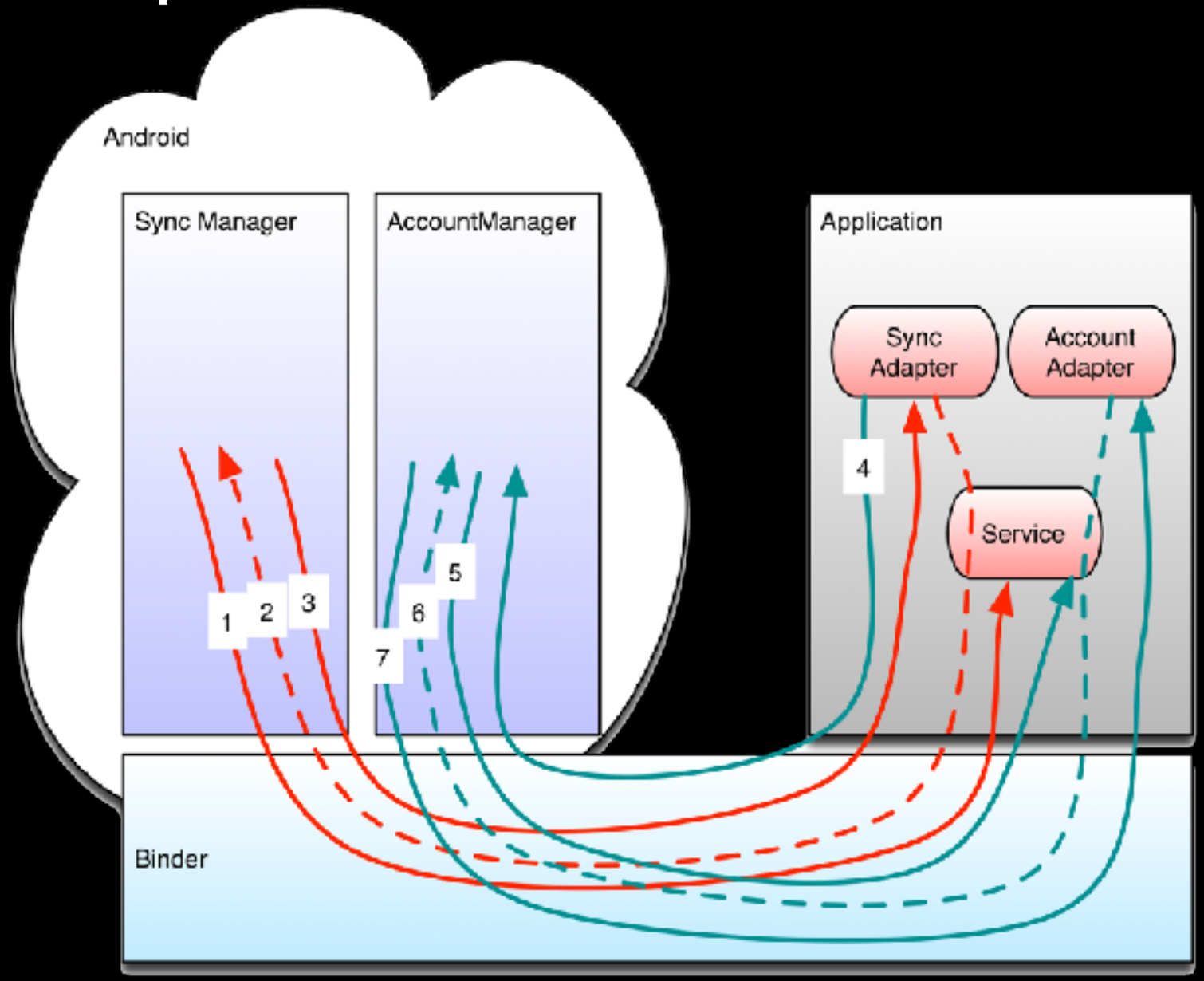
```
public class SyncService extends Service {
  private static final String ACTION_BIND_SYNC
    = "android.content.SyncAdapter";

  private SyncAdapter syncAdapter;

  @Override
  public void onCreate() {
    super.onCreate();
    syncAdapter = new SyncAdapter(getApplication(), true);
  }

  @Override
  public IBinder onBind(Intent intent) {
   if (ACTION_BIND_SYNC.equals(intent.getAction())) {
      return syncAdapter.getSyncAdapterBinder();
    }
    return null;
  }
}
```

```
public class SyncAdapter
  extends AbstractThreadedSyncAdapter
{

  public SyncAdapter(
    Context context,
    boolean autoInitialize)
  {
    super(context, autoInitialize);
  }

  @Override
  public void onPerformSync(
    Account account,
    Bundle bundle,
    String s,
    ContentProviderClient contentProviderClient,
    SyncResult syncResult)
  {
    doPeriodicTask(account, bundle)
  }
}
```

Sync Adapter: details

- The SyncManager binds your service, starting it if it is not already running, when a sync is required.
- Android holds a wakelock during execution
- The (considerable) complexity is mostly in Account Management

Sync Adapter: architecture



Sync Adapter: scheduling

- Explicitly (GCM): `requestSync`
- Periodically: `addPeriodicSync`
- Insanely elegant: `notifyChanged(uri, null, true);`
- When the radio wakes up to talk to the tower:

  `setSyncAutomatically`

SyncAdapter: hot

- You can probably avoid seeing threads at all
- Synchronizes two datasets: components are someone else's problem
- Runs exactly when it needs to
- Very power friendly

SyncAdapter: not

- Quite heavyweight
- Requires a Content Provider
- Requires an Account
- Limited scheduling options

SyncAdapter: hot

If you haven’t noticed…

Sync Adapters are actually pretty hot

your app as the VC, from MVC

SyncAdapter: btw…

The Realm Mobile Platform

is, essentially a simplified,

optimized SyncAdapter.

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty Power
TimerTask				
Looper/Handler				
Intent/Alarm Service				
SyncAdapter				
JobService				

JobScheduler

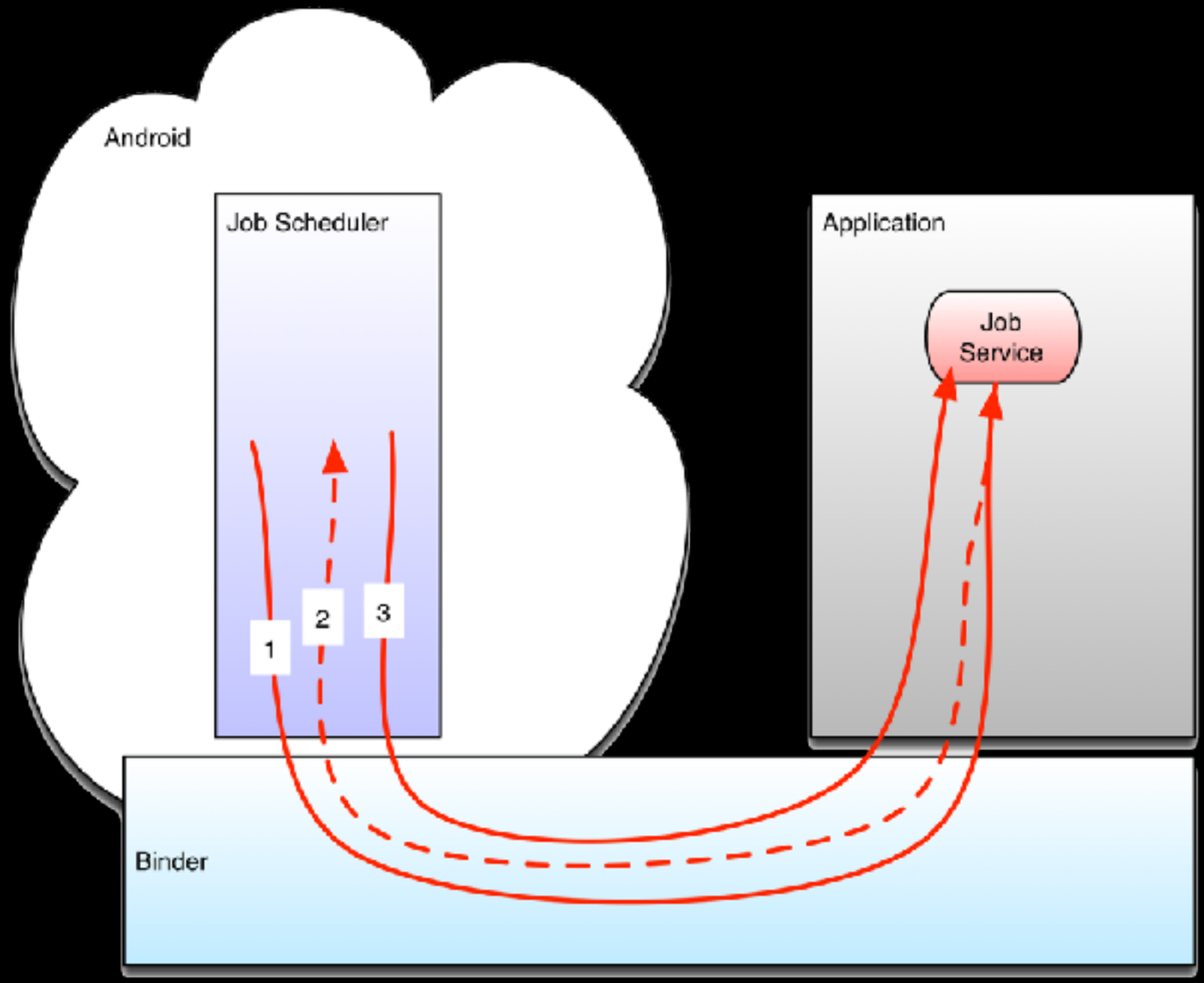
A Job is a task to be completed under specified conditions.

```
<service
  android:name=".svc.TaskScheduler"
  android:permission
    ="android.permission.BIND_JOB_SERVICE"/>
```

```
private static int jobId;

public void startPeriodicTask(
  long interval, int backoff, int network)
{
  PersistableBundle extras = new PersistableBundle();
  extras.putInt(PARAM_OP, OP_TASK_1);

  JobInfo job = new JobInfo.Builder(
    jobId++,
    new ComponentName(ctxt, TaskScheduler.class))
    .setExtras(extras)
    .setBackoffCriteria(
      backoff, JobInfo.BACKOFF_POLICY_EXPONENTIAL)
    .setPeriodic(interval)
    .setPersisted(true)
    .setRequiredNetworkType(network)
    .build();

  ((JobScheduler) ctxt.getSystemService(
    Context.JOB_SCHEDULER_SERVICE)).schedule(job);
}
```

```
public class TaskScheduler extends JobService {

@Override
public boolean onStartJob(JobParameters params) {
  PersistableBundle extras = params.getExtras();
  int op = (null == extras) ? 0 : extras.getInt(PARAM_OP);
  switch (op) {
    case OP_TASK_1:
    case OP_TASK_2:
      Message.obtain(handler, op, params).sendToTarget(); // !!!
      return true;
    default:
      Log.e(TAG, "Unexpected op: " + op);
    }
    return false;
}

@Override
public boolean onStopJob(JobParameters params) {
  // This is complicated...
  return false;
}
```

JobScheduler: architecture



Job Scheduler: details

- Service does not have to be exported
- Android holds a wakelock during execution
- Works with Accounts
- Persistent across reboot (requires ON_BOOT_COMPLETED)
- Jobs are executed on the UI thread!

Job Scheduler: scheduling

- Latency (jitter)
- Backoff time and strategy
- Connectivity (metered, not-metered, any)
- Deadline

Job Scheduler: stopping jobs

The documentation says:

You are solely responsible for the behavior of your application upon receipt of this message; your app will likely start to misbehave if you ignore it

Remember Thread.stop()?

Job Scheduler: hot

- Excellent fine grained scheduling: control over backoff, network cost, etc.
- Android holds the wakelock
- Very smart task clustering
- Simple to use

Job Scheduler: not

- You are on your own with concurrency
- A two-layered scheduling strategy is a long way from generalization of a one-layered strategy

Periodic task frameworks

	Thread safe	Lifecyle aware	Smart scheduling	Thrifty Power
TimerTask				
Looper/Handler				
Intent/Alarm Service				
SyncAdapter				
JobService				

So?

No silver bullets yet.

The JobSchedule is one more tool for one more set of special cases.

We have yet to achieve a unified theory of periodic tasks

Thank you!

blake.meike@gmail.com

twitter: @callmeike

blog: http://portabledroid.wordpress.com/