ADRIAN KREMSKI
Android Developer

JOBSCHEDULER TO THE RESCUE

In mobile development there are moments when an application just must be kept running regardless of the user interaction. Android 5.0 Lollipop brought us a new solution to this topic. Let's check it!

But wait a second... How did we manage to do this on pre-Lollipop devices?

Previously this could be achieved (to some extent) by using AlarmManager. However, this "Android cron" had some limitations:

- it didn't persist across reboots
- it didn't keep the device awake (solution may be also using https://github.com/commonsguy/cwac-wakeful)

- it didn't give control to execute our tasks only when needed criteria were met (e.g. wi-fi is available)

JOBSCHEDULER TO THE RESCUE!

In Lollipop devices JobScheduler comes to save us.
From https://developer.android.com/reference/android/app/job/JobScheduler.html:

"This is an API for scheduling various types of jobs against the framework that will be executed in your application's own process."

So let's get to business.

DECLARING JOBSERVICE

We will create our first JobService (this is the place where our jobs are going to be handled) with some basic logic.

```
1  @TargetApi(Build.VERSION_CODES.LOLLIPOP)
2  public class DownloadJobService extends JobService {
3
4      public static final String URLS_KEY = "urls_key";
5
6      private Download download;
7
8      @Override
9      public boolean onStartJob(JobParameters params) {
10         download = new Download(params);
11         download.start();
12         return true;
13     }
14
15     @Override
16     public boolean onStopJob(JobParameters params) {
17         download.interrupt();
18         Log.i("TASK : " + params.getJobId(), "Job stopped");
19         return false;
20     }
21
22     private class Download extends Thread {
23         private final JobParameters params;
24
25         Download(JobParameters params) {
26             this.params = params;
27         }
28
29         @Override
30         public void run() {
31             Log.i("TASK : " + params.getJobId(), "Job started");
32             download(params.getExtras().getStringArray(URLS_KEY), params);
33             jobFinished(params, false);
34             Log.i("TASK : " + params.getJobId(), "Job finished");
```

```
35          }
36
37          private void download(String[] urls, JobParameters parameters) {
38              for (String url : urls) {
39                  try {
40                      Thread.sleep(1000);
41                      Log.i("TASK : " + parameters.getJobId(), String.format("Downloaded %s!",
42                  } catch (InterruptedException e) {
43                      e.printStackTrace();
44                  }
45              }
46          }
47      }
48
49  }
```

- onStartJob(JobParameters params) is a method that the system triggers when you invoke your JobInfo task. Its parameter gives us the access to jobId (params.getJobId()) to identify our task and PersistableBundle with some extras (which we can pass through JobInfo during task invocation). For quick tasks onStartJob should return false, however, for longer ones (network operations, etc.), we have to call jobFinished(JobParameters params, boolean needsRescheduled) when our operations are done (in this case onStartJob should return true). An important note is that onStartJob is invoked in the main thread of the application (it's our responsibility to move the time consuming operations off thread).

- onStopJob(JobParameters params) will be called by the OS if the current work should be cancelled due to environment conditions (e.g. our job is restricted to be done in idle state, but our device is no longer in it). If the job is cancelled and we would like to retry it, the method should return true. `Rescheduling` can also be invoked by passing true to jobFinished method.

BUT HOW DOES THIS SCHEDULING MECHANISM ACTUALLY WORK?

There are two cases to consider:

1. - Our job is set to be invoked while the device is in idle state.
   In this case the work cancelled will be added to a queue and retried when the device goes back to idle.

2. - A canceled job that wasn't in idle mode will be retried after 30seconds with the BACKOFF_POLICY_EXPONENTIAL policy. However, if we reschedule it again (by passing true either to jobFinished or onStopJob method), then the time will grow exponentially with the formula 2 * (n-1) * t (n is the number of retries and t is 30s).

By default the retry policy is set to BACKOFF_POLICY_EXPONENTIAL but there is also the option BACKOFF_POLICY_LINEAR (with formula n*t) which you can set with setBackoffCriteria() on your JobInfo.Builder.

ANDROIDMANIFEST

As with all services on Android, we need to add our JobService to AndroidManifest, but in this casewe also have to add `android.permission.BIND_JOB_SERVICE` permission.
`If you skip it, the JobScheduler will be ignored by the OS.`

JOB INVOCATION

Now that we have the DownloadJobService done, let's invoke our first job. To do this, we will use JobInfo.Builder.

```
1  JobInfo jobInfo = new JobInfo.Builder(JOB_ID, new ComponentName(getBaseContext(), DownloadJo
2          .setBackoffCriteria(policy)
3          .setRequiresCharging(requiresCharging)
4          .setPersisted(persistsAfterReboot)
5          .setRequiresDeviceIdle(requiresIdle)
6          .setOverrideDeadline(deadline)
7          .setMinimumLatency(minimumLatency)
8          .setRequiredNetworkType(requiredNetworkType)
9          .setExtras(extras)
10         .build();
```

- `setBackoffCriteria (long initialBackoffMillis, int policy)` – change backo policy to BACKOFF_POLICY_LINEAR or BACKOFF_POLICY_EXPONENTIAL (default one) with initialBackoffMillis as the initial time interval to wait after a job failure
- `setRequiresCharging (boolean requiresCharging)` – the device needs to be charging to run this job (false by default)
- `setPersisted (boolean isPersisted)` – persistance across reboots. This call will only work if your application has RECEIVE_BOOT_COMPLETED permission in AndroidManifest (an exception will be thrown otherwise)
- `setOverrideDeadline (long maxExecutionDelayMillis)` – maximum delay before job invocation (even if other requirements are not met.). This works only for one-shot jobs (periodic ones will throw IllegalArgumentException on build)
- `setRequiresDeviceIdle (boolean requiresDeviceIdle)` – the device needs to be in idle state before running the next job (false by default)
- `setMinimumLatency (long minLatencyMillis)`– delay the job with given minLatencyMillis (periodic jobs will throw IllegalArgumentException on build)
- `setRequiredNetworkType (int networkType)` – a specific type of network your job

needs to have (NETWORK_TYPE_NONE by default)

- `setExtras(PersistableBundle extras)` – pass extra parameters with your job

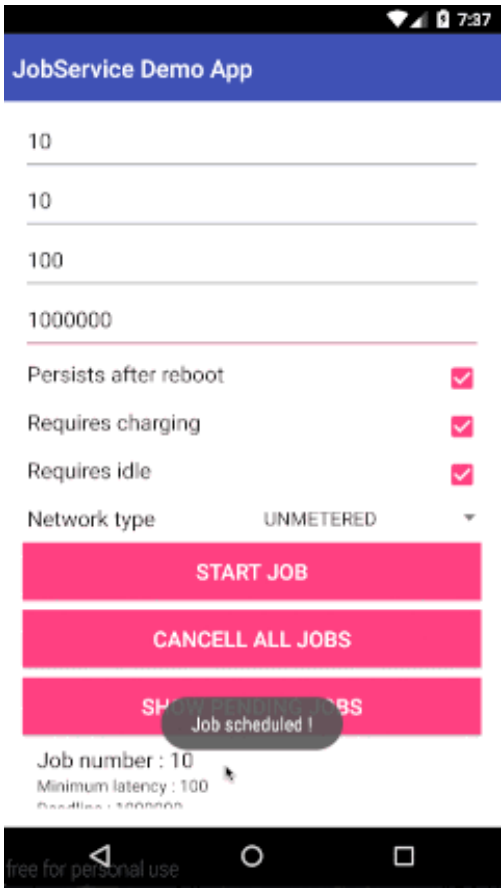
After preparing jobInfo, we are ready to go.

```
1 JobScheduler scheduler = (JobScheduler)getSystemService(Context.JOB_SCHEDULER_SERVICE);
2 int result = scheduler.schedule(jobInfo);
3 if (result == JobScheduler.RESULT_SUCCESS) {
4     Toast.makeText(this, "Job scheduled !", Toast.LENGTH_SHORT).show();
5 }
```

Other useful JobScheduler methods

- `getAllPendingJobs()` – returns list of jobs to be executed
- `cancelAll()` – cancels all jobs (this method will invoke JobService onStopJob(JobParameters params))
- `cancel(int jobId)` – cancels jobs with give jobId (this method will invoke JobService onStopJob(JobParameters params))

If you want to test the code in real application, check my Github repo and build it by yourself.



Thanks for reading !

Edit

COMMENTS

Leave a Reply

Logged in as Adrian Kremski. Log out?

Comment

POST COMMENT

MORE ARTICLES FROM SCHIBSTED TECH POLSKA

07. 05. 2015

8 cool Android libraries to speed up your development

Do you know what is among the best things of Android development? It is Java based and extremely popular. Our community is awesome and there are many great libraries popping out every day....

CONTINUE READING →

20. 04. 2015

10 tips for becoming a successful Android

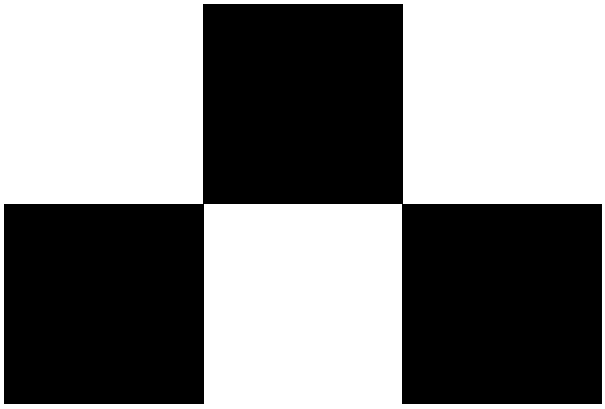
developer

Becoming a great Android developer is hard work and takes passion and patience. Here are my tips....

CONTINUE READING →

Get the latest job positions, hear about our events and learn how we work

SUBSCRIBE



Copyright © Schibsted Tech Polska