SensorEventListener in separate thread

This seems like a basic question, but after searching for a while and playing with it, I've come to the point where some help would be appreciated. I would like to have a SensorEventListener run in a separate thread from the UI, so that computations that need to occur when events come in won't slow down the UI.

My latest attempt looks like:

```
class SensorThread extends Thread {
    SensorManager mSensorManager;
    Sensor mSensor;

    public void run() {
        Log.d( "RunTag", Thread.currentThread().getName() ); // To display thread
        mSensorManager = (SensorManager)getSystemService( SENSOR_SERVICE  );
        mSensor = mSensorManager.getDefaultSensor( Sensor.TYPE_ACCELEROMETER );
        MySensorListener msl = new MySensorListener();
        mSensorManager.registerListener(msl, mSensor, SensorManager.SENSOR_DELAY_UI
);
    }

    private class MySensorListener implements SensorEventListener {
        public void onAccuracyChanged (Sensor sensor, int accuracy) {}
        public void onSensorChanged(SensorEvent sensorEvent) {
            Log.d( "ListenerTag", Thread.currentThread().getName() ); // To display
thread
        }
    }
```

edited Jul 20 '10 at 6:50

MatrixFrog
15.5k 9 47 81

asked Jul 20 '10 at 3:33

Thinman
66 1 3

possible duplicate of Acclerometer Sensor in Separate Thread – Alex Cohn Oct 3 '13 at 9:52

5 Answers

You can receive sensor event in a background thread. Instead of

```
mSensorManager.registerListener(msl, mSensor, SensorManager.SENSOR_DELAY_UI );
```

you can declare it with a handler referring to a secondary thread. The run loop of a thread looks like this:

```
...
run {
    Looper.prepare;
    Handler handler = new Handler();
    ...
    mSensorManager.registerListener(msl, mSensor, SensorManager.SENSOR_DELAY_UI,
handler );
    Looper.loop();
}
...
```

answered Jul 20 '11 at 9:37

Sney
1,246 2 24 46

public void run(){... and Looper.prepare(); – Slackware Jan 6 '14 at 18:01

Or use android.os.HandlerThread and create a Handler: new Handler(yourHandlerThread.getLooper()) – Tore Rudberg Jan 15 '16 at 9:00

A little late, but if others still want to know, here is a good way to achieve this. As always when multithreading, make sure you know what you are doing and take the time to so it right, to avoid those weird errors. Have fun!

Class members:

```
private HandlerThread mSensorThread;
private Handler mSensorHandler;
```

in OnCreate or when registering:

```
mSensorThread = new HandlerThread("Sensor thread", Thread.MAX_PRIORITY);
mSensorThread.start();
mSensorHandler = new Handler(mSensorThread.getLooper()) //Blocks until looper is
prepared, which is fairly quick
yourSensorManager.registerListener(yourListener, yourSensor, interval,
mSensorHandler);
```

When unregistering, also do:

```
mSensorThread.quitSafely();
```

answered Jan 15 '16 at 9:37


Tore Rudberg
1,034 11 15

perhaps the best answer – Taranfx Mar 13 '17 at 18:55

`Thread.MAX_PRIORITY` should be changed to `Process.THREAD_PRIORITY_MORE_FAVORABLE` as recommended in HandlerThread Android Developers "[...] The value supplied must be from Process and not from java.lang.Thread." – Nuno Henriques Dec 17 '17 at 18:05

It looks like the SensorManager is actually responsible for calling the onSensorChanged

to make any difference. The easiest thing to do is probably to make onSensorChanged return instantly, by delegating all the heavy lifting to a separate thread. Or perhaps to an ASyncTask, which seems to be the official "right way" of doing such things.

answered Jul 20 '10 at 6:50

MatrixFrog
15.5k 9 47 81

Thanks for the idea. I come from the 8-bit micro world, so I may need to adjust my thinking some. Allocating new tasks every time the sensor changes (5-50x a second, depending on sample rate) seems like a waste. – Thinman Jul 20 '10 at 20:45

2 Well, maybe it would only allocate the task in certain cases. For example, maybe it stores the sensor reading in memory, and then checks in the onSensorChanged method whether the reading has changed more than some threshold. If it has, THEN it starts up the ASyncTask. I'm really just getting started with Android though, so I don't really know what will work best. – MatrixFrog Jul 21 '10 at 4:28

OnSensorChanged should really run in ASyncTask - even if your main class is a background service(like in my case) it may be that if don't return fast enough from the method, the next call of SensorManager cannot be made... – Radu Aug 16 '12 at 17:21

Get the handler of the thread and register the listener on that thread. For example:

```
public void run() {
    Sensor sensor = this.sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
    Looper.prepare();
    Handler gHandler = new Handler();
    this.sensorManager.registerListener(gravitySensorEventListener, sensor,
SensorManager.SENSOR_DELAY_NORMAL, gHandler);
    Looper.loop();
}
```

hopes this would help you.

edited Jun 19 '16 at 0:47

Benoît Latinier
1,632 2 16 24

answered Jun 18 '16 at 20:51

Sayyed Erfanian
99 1 1 11

listener as third argument.

```
public void run() {
    Log.d( "RunTag", Thread.currentThread().getName() ); // To display thread
    mSensorManager = (SensorManager)getSystemService( SENSOR_SERVICE  );
    mSensor = mSensorManager.getDefaultSensor( Sensor.TYPE_ACCELEROMETER );
    MySensorListener msl = new MySensorListener();
    Looper.perpare;
    Handler hndlr = new Handler();

    mSensorManager.registerListener(msl, mSensor,
     SensorManager.SENSOR_DELAY_UI, hndlr);
    Looper.loop;

    }
```

edited Mar 17 '16 at 6:25

answered Mar 17 '16 at 6:00

Baryal

31 3