

Android (/tags/#Android)

PowerManager (/tags/#PowerManager)

Doze (/tags/#Doze)

Android电源管理之Doze模式专题系列（六）

状态切换剖析之SENSING-->LOCATION

Posted by Cheson on March 15, 2017

状态切换剖析之IDLE_PENDING-->SENSING (https://chendongqi.github.io/blog/2017/03/04/pm_doze_pending_to_sensing/)的最后部分提到了从SENSING状态向LOCATION切换的一个入口，这篇将接上文来继续阐述SENSING状态是如何切换到LOCATION的。

在SENSING状态下最后所做的一件事就是启动了MotionDector来监测是否设备是否有位移发生，监测的结果会通过回调函数返回。如果状态是RESULT_STATIONARY并且当前状态为SENSING那么将调用stepIdleStateLocked来开始切换。

```
@Override
public void onAnyMotionResult(int result) {
    if (DEBUG) Slog.d(TAG, "onAnyMotionResult(" + result + ")");
    if (result == AnyMotionDetector.RESULT_MOVED) {
        if (DEBUG) Slog.d(TAG, "RESULT_MOVED received.");
        synchronized (this) {
            handleMotionDetectedLocked(mConstants.INACTIVE_TIMEOUT, "sense_motion");
        }
    } else if (result == AnyMotionDetector.RESULT_STATIONARY) {
        if (DEBUG) Slog.d(TAG, "RESULT_STATIONARY received.");
        if (mState == STATE_SENSING) {
            // If we are currently sensing, it is time to move to locating.
            synchronized (this) {
                mNotMoving = true;
                stepIdleStateLocked();
            }
        } else if (mState == STATE_LOCATING) {
            // If we are currently locating, note that we are not moving and step
            // if we have located the position.
            synchronized (this) {
                mNotMoving = true;
                if (mLocated) {
                    stepIdleStateLocked();
                }
            }
        }
    }
}
```

在stepIdleStateLocked方法中判断当前为SENSING的状态，然后做了几件事：1）修改当前状态；2）设置Alarm；3）设置位置监听

```
case STATE_SENSING:
    mState = STATE_LOCATING;
    if (DEBUG) Slog.d(TAG, "Moved from STATE_SENSING to STATE_LOCATING.");
    EventLogTags.writeDeviceIdle(mState, "step");
    cancelSensingAlarmLocked();
    scheduleSensingAlarmLocked(mConstants.LOCATING_TIMEOUT);
    mLocating = true;
    locationManager.requestLocationUpdates(mLocationRequest, mGenericLocationListener,
        mHandler.getLooper());
    if (mLocationManager.getProvider(LocationManager.GPS_PROVIDER) != null) {
        mHaveGps = true;
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 5,
            mGpsLocationListener, mHandler.getLooper());
    } else {
        mHaveGps = false;
    }
    break;
```

第一点，也就是简单的将变量mState的值赋值为了STATE_LOCATING。第二点做的事一如前面几种状态中做过的一样，设置一个Alarm来作为下个状态切换的触发器。在设置之前首先要cancel以前的Alarm，这个被取消的广播类型就是ACTION_STEP_IDLE_STATE，也就是触发状态切换的广播。然后调用scheduleSensingAlarmLocked(mConstants.LOCATING_TIMEOUT)方法来重新设置Alarm，这个Alarm的deley时间就是LOCATING_TIMEOUT。

```
LOCATING_TIMEOUT = mParser.getLong(KEY_LOCATING_TIMEOUT,
    !DEBUG ? 30 * 1000L : 15 * 1000L);
```

默认DEBUG为false，所以这个timeout默认为30秒，后面也会介绍到在这30秒的时间内用来检测设备的运动状态而在scheduleSensingAlarmLocked方法中看到又进行了一次取消alarm的操作，这里是不是多余的呢？之前刚进行过一次取消。

```
void scheduleSensingAlarmLocked(long delay) {
    if (DEBUG) Slog.d(TAG, "scheduleSensingAlarmLocked(" + delay + ")");
    cancelSensingAlarmLocked();
    mNextAlarmTime = SystemClock.elapsedRealtime() + delay;
    mAlarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
        mNextAlarmTime, mSensingAlarmIntent);
    mSensing = true;
}
```

第三点就是设置位置监听，首先将mLocating标志位置为true，也就是标示注册了位置监听服务，在后面的取消位置监听时作为标志位来判断。然后通过LocationManager了请求了位置监听服务，先是申请了通过手机本身传感器精度为ACCURACY_FINE的定位服务

```
mLocationRequest = new LocationRequest()
    .setQuality(LocationRequest.ACCURACY_FINE)
    .setInterval(0)
    .setFastestInterval(0)
    .setNumUpdates(1);
```

而监听器的定义如下

```
private final LocationListener mGenericLocationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        synchronized (DeviceIdleController.this) {
            receivedGenericLocationLocked(location);
        }
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
    }

    @Override
    public void onProviderEnabled(String provider) {
    }

    @Override
    public void onProviderDisabled(String provider) {
    }
};
```

在触发位置变化时去调用receivedGenericLocationLocked(location)

```
void receivedGenericLocationLocked(Location location) {
    if (mState != STATE_LOCATING) {
        cancelLocatingLocked();
        return;
    }
    if (DEBUG) Slog.d(TAG, "Generic location: " + location);
    mLastGenericLocation = new Location(location);
    if (location.getAccuracy() > mConstants.LOCATION_ACCURACY && mHaveGps) {
        return;
    }
    mLocated = true;
    if (mNotMoving) {
        stepIdleStateLocked();
    }
}
```

如果开启了Gps的定位监测，并且精度大于LOCATION_ACCURACY，那么这里的监测就忽略(用更精确的GPS数据)，用Gps的监测数据，在后面会讲到。否则,如果监测数据为Not move，那么就调用StepIdleStateLocked来进行下一步。在此同时，也去检测了是否支持GPS

```
mLocationManager.requestLocationUpdates(mLocationRequest, mGenericLocationListener,
                                         mHandler.getLooper());
if (mLocationManager.getProvider(LocationManager.GPS_PROVIDER) != null) {
    mHaveGps = true;
    mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 5,
                                             mGpsLocationListener, mHandler.getLooper());
} else {
    mHaveGps = false;
}
```

如果支持的话就申请注册GPS的位置监测服务，以提供准确度。同时前面ACCURACY_FINE的定位服务实际上就被架空了。来看下这个监听器的定义。

```
private final LocationListener mGpsLocationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        synchronized (DeviceIdleController.this) {
            receivedGpsLocationLocked(location);
        }
    }
}
```

重点还是onLocationChanged时调用的receivedGpsLocationLocked方法，和receivedGenericLocationLocked类似

```
void receivedGpsLocationLocked(Location location) {
    if (mState != STATE_LOCATING) {
        cancelLocatingLocked();
        return;
    }
    if (DEBUG) Slog.d(TAG, "GPS location: " + location);
    mLastGpsLocation = new Location(location);
    if (location.getAccuracy() > mConstants.LOCATION_ACCURACY) {
        return;
    }
    mLocated = true;
    if (mNotMoving) {
        stepIdleStateLocked();
    }
}
```

和前面的类似，在结果为not move的时候调用stepIdleStateLocked来进入下一步操作。至此从SENSING状态切换到LOCATION状态的动作就完成了。

PREVIOUS

ANDROID PERFORMANCE PATTERNS
——MEMORY PERFORMANCE (2017/03/15
/ANDROID_PERF_PATTERNS_MEMORY/)

NEXT

吉他谱——赵雷_理想 (2017/03/16
/GUITAR_ZHAOLEI_LIXIANG/)

FEATURED TAGS (/tags/)

- 前端 (/tags/#前端)
- Android (/tags/#Android)
- frameworks (/tags/#frameworks)
- AlarmManager (/tags/#AlarmManager)
- Performance (/tags/#Performance)
- systrace (/tags/#systrace)
- PowerManager (/tags/#PowerManager)
- Wakelock (/tags/#Wakelock)
- Guitar (/tags/#Guitar)
- 民谣 (/tags/#民谣)
- 赵雷 (/tags/#赵雷)
- Doze (/tags/#Doze)
- Android Performance Patterns (/tags/#Android Performance Patterns)

FRIENDS

待遇见志同道合的你 (https://github.com) 小明 (http://www.betterming.cn)

(https://twitter.com/chendongqi)

(https://www.zhihu.com/people/chendongqi)

(http://weibo.com/chendongqi)

(https://www.facebook.com/chendongqi)

(https://github.com/chendongqi)

(https://www.linkedin.com/in/firstname-lastname-idxxxx)