Android 6.0省电模式研究（一）

Jul 13, 2016 | 深入调研 | 149 Hits

最近做省电模式的深度调研，记录一下。

问题背景

最近发现在Android 6.0以上的系统，有个比较恶心的东东，那就是省电模式。对用户比较好，但是对开发者来说相当不方便。下面描述一下现象：
条件：省电模式开启
现象：如果Activity在前台运行的时候，Activity所在的Application下的service，均可以发起网络请求（通过识别Application的id）；如果application没有任何Activity在前台运行，则其下的任何service被拒绝了网络连接。

问题官方解决办法

方法一

在API 23下PowerManager增加了函数isPowerSaveMode，判断是否处于省电模式；还有函数isIgnoringBatteryOptimizations，判断是否在省电白名单中。为了兼容全部Api，因此我这边用反射修改了一下，思路一样的：

```
if (Build.VERSION.SDK_INT >= 23) {
    try {
        // 是否是省电模式
        PowerManager pm = (PowerManager) OpenWnnSimeji.this.getSystemService(POWER_SERVICE);
        Class<?> cls = pm.getClass();
        Method method1 = cls.getMethod("isPowerSaveMode", new Class<?>[]{});
        Method method2 = cls.getMethod("isIgnoringBatteryOptimizations", String.class);
        boolean isSavedMode = (Boolean) method1.invoke(pm, new Object[]{});
        boolean isWhileList = (Boolean) method2.invoke(pm, getPackageName());
        Log.i(TAG, "isSavedMode : " + isSavedMode + ", isWhileList :" + isWhileList);
        if (isSavedMode && !isWhileList) {
            // 跳转到系统省电白名单设置页面
            try {
                Intent ignore = new Intent("android.settings.IGNORE_BATTERY_OPTIMIZATION_SETTINGS");
                ignore.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(ignore);
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

这个方法好处就是不用在App增加任何权限，但是跳到设置页面，用户完全懵了，不知道怎么设置，交互和体验不好。

方法二

和方法一类似，但是这个方法要增加一个权限，用户只需要确认是否把当前app加到白名单。
增加的权限：

```
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
```

代码上和方法一很像，只有在里面跳转的时候不一样

```
if (Build.VERSION.SDK_INT >= 23) {
    try {
        // 是否是省电模式
        PowerManager pm = (PowerManager) OpenWnnSimeji.this.getSystemService(POWER_SERVICE);
        Class<?> cls = pm.getClass();
        Method method1 = cls.getMethod("isPowerSaveMode", new Class<?>[]{});
        Method method2 = cls.getMethod("isIgnoringBatteryOptimizations", String.class);
        boolean isSavedMode = (Boolean) method1.invoke(pm, new Object[]{});
        boolean isWhileList = (Boolean) method2.invoke(pm, getPackageName());
        Log.i(TAG, "isSavedMode : " + isSavedMode + ", isWhileList :" + isWhileList);
        if (isSavedMode && !isWhileList) {
            // 弹窗是否把当前App添加到白名单
            try {
                Intent ignore = new Intent("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
                ignore.setData(Uri.parse("package:" + getPackageName()));
                ignore.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(ignore);
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

为了减少升级确认带来的伤害，在添加权限的时候，可以使用

```
<uses-permission-sdk-23 android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
```

方法三

在Android API 21引入了一个东东JobScheduler，这个是系统调度服务。可以用来解决问题。
当然使用上也是挺麻烦的。核心代码如下：

```
JobInfo netJob = new JobInfo.Builder(
        JOB_ID, /** 自己定义 */
        new ComponentName(content, MyJobService.class))
                .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
                .setRequiresCharging(true) /** 注意，这一行一点要写，否则上面设置无法生效，看源码可以看到，这个是f
                .build();
/** 低API 就是用：#JOB_SCHEDULER_SERVICE ("taskmanager") */
JobScheduler jobScheduler = (JobScheduler) context.getSystemService(Context.JOB_SCHEDULER_SERVICE);
jobScheduler.schedule(netJob);

/** MyJobService 是什么鬼 */
public class MyJobService extends JobService {
        @Override
        public boolean onStartJob(JobParameters params) {
                /** 我们看下面的源码发现，这个函数居然是在UI线程执行，这个一定要注意 */
                return true;
        }

        @Override
        public boolean onStopJob(JobParameters params) {
                return true;
        }
}

/** JobService 源码 */
/** Binder for this service. */
IJobService mBinder = new IJobService.Stub() {
        @Override
```

```
        public void startJob(JobParameters jobParams) {
            ensureHandler();
            Message m = Message.obtain(mHandler, MSG_EXECUTE_JOB, jobParams);
            m.sendToTarget();
        }
        @Override
        public void stopJob(JobParameters jobParams) {
            ensureHandler();
            Message m = Message.obtain(mHandler, MSG_STOP_JOB, jobParams);
            m.sendToTarget();
        }
    };

    /** @hide */
    void ensureHandler() {
        synchronized (mHandlerLock) {
            if (mHandler == null) {
                mHandler = new JobHandler(getMainLooper());
            }
        }
    }
```

看上去好像还可以接受，这个不需要用户干涉，自己可以执行，但是，还是要求添加权限：

```
<uses-permission android:name="android.permission.BIND_JOB_SERVICE" />
```

这个最大缺点不是我们执行的问题，而是这个是系统service调度的，如果多任务执行的时候，对一些时效要求高，请求频繁的App，如在线播放器，输入法等，就不太合适。

问题描述

所以我们很快就有疑问，有啥办法可以实现高频网络请求吗？而且尽可能不要添加权限，尽可能不要用户干预。
排了个优先级，调整了目标为：

- 实现高频网络
- 尽可能不要用户干预
- 尽可能不要添加权限（可以放宽）

带着这个问题，开展了源码阅读之旅。

Android 省电模式 分享到

Android 6.0省电模式研究（二） 输入法开发的一些误区