**电量优化：**

而开发者在电量消耗方面也起到了推波助澜的作用：相比于卡顿、内存泄漏等问题，开发者对电量消耗的重视程度极低；

开发者和QA工作时，会习惯性的连接电脑或者电源随时充电，电量问题根本暴露不出来；

开发者和QA的工作重点主要放在业务功能完成度上，类似卡顿、内存泄漏等性能问题直到暴露出来才会去解决，更何况不影响开发者和QA的电量消耗。

然而开发者的不关注并不代表用户的忽视，Android设备用户会普遍装载管家类App，通过这些管家App，用户可以轻松找到那些“电池杀手”应用，然后就是删除————》差评————》转向竞争对手应用一条龙。因此对于开发者而言要尽量少用电量，合理使用电池。本节就来一起探索既可以省电，又不影响用户体验的方法。

2、 电量测试

Android4.1版本之后在系统增加了battery info模块，记录一定时间周期内整机及单个App的电量消耗。

2.1 注册广播

ACTION\_BATTERY\_CHANGED

IntentFilter filter = new IntentFilter();

filter.addAction(Intent.ACTION\_BATTERY\_CHANGED);

registerReceiver(filter,receiver);

然后就可以获取电池电量、充电状态、电池状态等信息。具体参考BatteryManager。

缺点：

获取到的是手机整体的耗电量，而不是特定App的耗电量；

实时性差，精度较低，只能接受被动通知电量余量以及跳变。

2.2 Battery Historian

最强大、最推荐的工具：Battery Historian是Android5.0之后Google开源的一款用于检测与电池有关的信息和事件的工具，从设备中收集电池数据，然后使用Battery Historian可以可视化分析相关指标如耗电比例、Wifi、蜂窝数据量、WakeLock唤醒次数。随着Android6.0更新了Battery Historian 2.0加入引起手机状态变化的应用。

通过Battery Historian可以方便的看到各耗电模块随着时间的耗电情况：包含操作类型、执行时间、对应App等；还可以进行筛选特定的App，给出一个总结性的说明，包括：Network Information、 Syncs、WakeLock、Services、Process info、Scheduled Job、Sensor Use等，查看每一个模块的总结，可以看出来每一项的耗时以及执行次数。当发现异常的时候可以针对性的进行排查。总之：Battery Historian真的很强大。

adb命令导出电量信息：

adb shell dumpsys batterystats --reset（Android4.1到4.3 adb shell dumpsys batteryinfo）

adb bugreport > bugreport.txt(Android7.0以上 adb bugreport bugreport.zip)

安装Battery Historian后打开：http: //localhost:9999/， 上传bugreport.txt文件开始分析，下图分析360手机助手为例；

11点44分06秒和11点55分10秒发生两次JobScheduler操作，图有木有很像TraceView

单独查看360手机助手，此处显示WakeLock的使用

可以看出：360手机助手使用WakeLock的场景有：推送、定时任务、利用系统账号同步、服务等。

悄悄的告诉你：360手机助手相比于一般应用耗电的场景更多哦，当然对于一个超级App，也不能过多要求。

安装过程可以参考Github：battery-historian。备注：我使用Docker的方式并没有执行成功，通过Go的方式完成的。

3、 电量优化

Android系统上App的电量消耗主要由cpu、wakelock、数据传输（流量和wifi）、wifi运行、gps、other senior组成，而耗电异常也是由于这几个模块的使用不当。

3.1 CPU时间片优化

当检测到CPU时间片消耗异常时，需要使用TraceView，获取进程执行信息，定位CPU占用率异常的问题，关于CPU的使用可以参照《Android性能优化（一）之启动加速35%

》一文。

3.2 网络传输

通常情况下，使用3G移动网络传输数据，电量的消耗有三种状态：

Full power: 能量最高的状态，移动网络连接被激活，允许设备以最大的传输速率进行操作。

Low power: 一种中间状态，对电量的消耗差不多是Full power状态下的50%。

Standby: 最低的状态，没有数据连接需要传输，电量消耗最少。

3.2.1 数据压缩

通过数据压缩等方式缩减传输时间，降低电量消耗，此章节可以参考《Android 性能优化（八）之网络优化》。

3.2.2 选择更快的传输方式

虽然3G芯片比Wifi芯片耗电低，但Wifi的速率可以让数据在较短时间内完成传输，从而降低电量消耗。

3.2.3 请求集中发送

分析和统计之类的非重要操作，可以在合适状态（电量充足或Wifi状态）下发送。参见3.6节JobScheduler。

3.2.4 无网状态避免网络请求

之前在网络优化的文章里写过，网络请求失败之后的重试机制，但是要注意这个重试是在有网状态下的重试。否则无网状态下重试不会请求成功，只会消耗电量。尤其是与AlarmManager或者WakeLock连用的场景下，耗电量会更多。

3.3 GPS

定位是App中常用的功能，但是定位不能千篇一律，不同的场景以及不同类型的App对定位更加需要个性化的区分。

3.3.1 选择合适的Location Provider

Android系统支持多个Location Provider：

GPS\_PROVIDER:

GPS定位，利用GPS芯片通过卫星获得自己的位置信息。定位精准度高，一般在10米左右，耗电量大；但是在室内，GPS定位基本没用。

NETWORK\_PROVIDER：

网络定位，利用手机基站和WIFI节点的地址来大致定位位置，这种定位方式取决于服务器，即取决于将基站或WIF节点信息翻译成位置信息的服务器的能力。

PASSIVE\_PROVIDER:

被动定位，就是用现成的，当其他应用使用定位更新了定位信息，系统会保存下来，该应用接收到消息后直接读取就可以了。比如如果系统中已经安装了百度地图，高德地图(室内可以实现精确定位)，你只要使用它们定位过后，再使用这种方法在你的程序肯定是可以拿到比较精确的定位信息。

使用Criteria，设置合适的模式、功耗、海拔、速度等需求，系统会返回合适的Location Provider。

例如你的App只是需要一个粗略的定位那么就不需要使用GPS进行定位，既耗费电量，定位的耗时也久。

3.3.2 及时注销定位监听

在获取到定位之后或者程序处于后台时，注销定位监听，此时监听GPS传感器相当于执行no-op（无操作指令），用户不会有感知但是却耗电。

public void onPause() {

super.onPause();

locationManager.removeListener(locationListener);

}

public void onResume(){

super.onResume();

locationManager.requestLocationUpdates(locationManager.getBestProvider(criteria, true),6000,100,locationListener);

}

3.3.3 多模块使用定位尽量复用

多个模块使用定位，尽量复用上一次的结果，而不是都重新走定位的过程，节省电量损耗；例如：在应用启动的时候获取一次定位，保存结果，之后再用到定位的地方都直接去取。

3.4 谨慎使用WakeLock

Android为了节省电量，会在用户无操作一段时间之后进入休眠状态。Wake Lock是一种锁的机制，只要有人拿着这个锁，系统就无法进入休眠。一些App为了能在后台持续做事情，就会持有一个WakeLock，那么手机就不会进入休眠状态，App要做的事情能做了，但是也更加耗电。

App在前台不要申请WakeLock，此时无需申请，申请的话会计算到应用电量消耗；

App在后台由于业务需要必须要申请WakeLock时使用带有超时参数的方法，防止由于忘记或者异常情况下没有释放；

App申请使用WakeLock，任务结束之后及时释放，让系统再次进入休眠状态。

PowerManager pm = (PowerManager)mContext.getSystemService(Context.POWER\_SERVICE);

PowerManager.WakeLock wl = pm.newWakeLock(PowerManager.SCREEN\_DIM\_WAKE\_LOCK| PowerManager.ON\_AFTER\_RELEASE,TAG);

wl.acquire(TIMEOUT);// 使用带有超时参数的acquire方法

// ... do work...

wl.release();

备注：如果只是需要屏幕常亮的话，可以使用FLAG\_KEEP\_SCREEN\_ON，无需考虑释放WakeLock的问题。

3.5 传感器使用

使用传感器，选择合适的采样率，越高的采样率类型则越费电；

SENSOR\_DELAY\_NOMAL (200000微秒)

SENSOR\_DELAY\_UI (60000微秒)

SENSOR\_DELAY\_GAME (20000微秒)

SENSOR\_DELAY\_FASTEST (0微秒)

在后台时注意及时注销传感器监听；

3.6 JobScheduler

使用JobScheduler，一些任务通过JobScheduler来触发，例如可推迟的网络请求、下载、GPS等，可以在特定场景：连接Wifi、连接电源等场景触发。既完成了任务，也无需考虑由于一些任务导致的电量消耗。

4、 后记

4.1 电量优化的一般套路

在设置-电量里查看App的耗电情况；

使用Battery Historian进行分析，这是分析里最重要的一步；

针对分析结果，参照第三章节的优化方式进行优化。

4.2 Android系统费电吗？

一直有一种传言：Android系统比较费电，然而真相不是这样，请不要把锅甩给Android系统：

原生的Android手机其实并不耗电，不安装App的Android手机放置一周仍然是电量充足，而且对功耗的控制在Android每次版本更新都会有所补强。

耗电的原因在于手机ROM以及安装的软件，手机ROM会针对原生的Android做各种各样的定制（免费赠送各种“亲情软件”，各种系统级应用）。安装软件的开发者不考虑电量损耗，以及都希望千方百计占用系统资源（例如保活、互拉）等。

电量优化可以说是开发者和QA最不关注的一个方面了，但是如果任而由之，变成“电量杀手”不仅仅是伤害用户的体验，也是对自己的放纵。性能问题不仅仅在于发现之后的优化更改，更在平时的防微杜渐。

**布局优化：**

include

merge

ViewStub

工具Hierarchy Viewer

布局优化工具（Lint）

Lint规则如下：

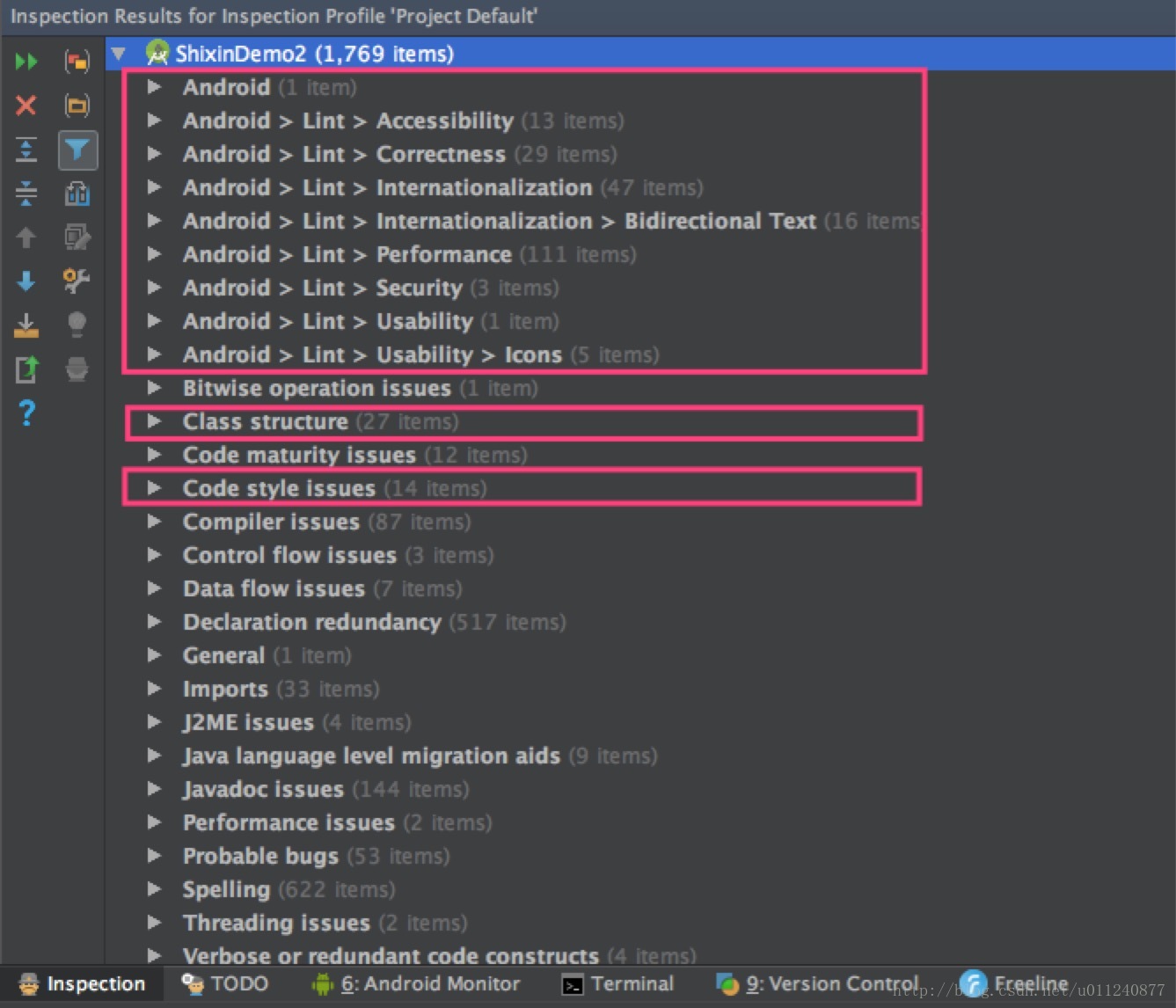
1、使用组合控件 ---包含了一个 ImageView以及一个 TextView控件的 LinearLayout如果能够作为一个组合控件将会被更有效的处理。

2、合并作为根节点的帧布局(Framelayout) ----如果一个帧布局时布局文件中的根节点，而且它没有背景图片或者padding等，更有效的方式是使用<merge />标签替换该< Framelayout/>标签。

3、无用的子节点-----通常来说如果一个布局控件没有子视图或者背景图片，那么该布局控件时可以被移除 (由于它处于 invisible状态)。

4、无用的父节点 ----- 如果一个父视图即有子视图，但没有兄弟视图节点，该视图不是ScrollView控件或者根节点，并且它没有背景图片，也是可以被移除的，移除之后，该父视图的所有子视图都直接迁移至之前父视图的布局层次。同样能够使解析布局以及布局层次更有效。

5、过深的布局层次 ----内嵌过多的布局总是低效率地。考虑使用一些扁平的布局控件，例如 RelativeLayout、GridLayout，来改善布局过程。默认最大的布局深度为10。



我们主要关注红框内的警告，先来看看我的代码 Performance 有什么问题：