# APP耗电评判方案

[APP耗电评判方案 1](#_Toc14935)

[Framework层 2](#_Toc9870)

[App占用电流数据 2](#_Toc1833)

[计算耗电实现 2](#_Toc28101)

[硬件单位时间耗电数据 2](#_Toc30740)

[Profile常量数据集合 3](#_Toc22640)

[App耗电量统计 4](#_Toc19573)

[共测量耗电值(应用级) 13](#_Toc14412)

[硬件耗电量统计 14](#_Toc18304)

[测量耗电值(device级) 16](#_Toc25193)

[耗电信息汇总 17](#_Toc13834)

[参考 18](#_Toc5188)

[Q1解决方案 19](#_Toc23474)

[Framework层 19](#_Toc22218)

[App层 19](#_Toc19020)

[Q2解决方案 20](#_Toc30357)

[各组件耗电排行 20](#_Toc32192)

[算法模拟 20](#_Toc8920)

[Q3解决方案 23](#_Toc10169)

[Reference 24](#_Toc4304)

[Attachment 25](#_Toc12748)

**APP耗电评判方案**

Questions

如何判断一个APP耗电量，并且决定APP等级，属于正常耗电，高耗电，异常耗电

思路：

检索APP耗电记录，并统计APP每部分耗电情况，如CPU,WIFI,GPS,SCREEN等

根据每部分耗电量，并做出统计，给定阈值，每部分占用比例进行耗电打分，并给出正常标准，超标情况下属于高耗电，在超过最大值时判定为异常耗电

难点：

1.如何统计出APP所有部件的耗电信息？

2.对每部分耗电信息值计算的时候，各部分占比多少来确定APP耗电值？

3.给定的正常耗电标准值为多少，高耗电标准值是多少，异常耗电标准值是多少？

Q1研究方案

Framework层

系统对APP做出了电量的详细记录，需要对其读取并分析

核心类：PowerUsageSummary.java

Displays a list of apps and subsystems that consume power, ordered by how much power was consumed since the last time it was unplugged

显示子系统和app耗电量，并根据上次拔下电源到目前的耗电量

源码地址：(基于Android 7.0)

<http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageSummary.java>

<http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageBase.java>

[mStatsHelper](http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageBase.java" \l "mStatsHelper).[refreshStats](http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageBase.java" \l "refreshStats)([BatteryStats](http://androidxref.com/7.0.0_r1/s?defs=BatteryStats&project=packages).[STATS\_SINCE\_CHARGED](http://androidxref.com/7.0.0_r1/s?defs=STATS_SINCE_CHARGED&project=packages), [mUm](http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageBase.java" \l "mUm).[getUserProfiles](http://androidxref.com/7.0.0_r1/s?defs=getUserProfiles&project=packages)()); //更新App耗电记录

<http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/BatteryStatsHelper.java>

[List](http://androidxref.com/4.0.3_r1/s?defs=List&project=packages)<[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)> **[mUsageList](http://androidxref.com/4.0.3_r1/s?refs=mUsageList&project=packages)** = **new** [ArrayList](http://androidxref.com/4.0.3_r1/s?defs=ArrayList&project=packages)<[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)>(); //App耗电集合

[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages):单个APP各部件详细耗电信息

App占用电流数据

[BatteryStatsHelper](http://androidxref.com/7.0.0_r1/s?refs=BatteryStatsHelper&project=frameworks) 获取不同app使用电量，读取/data/system/batterystats.bin文件，该文件记录Device使用电量标准值

计算耗电实现

**[processAppUsage](http://androidxref.com/4.0.3_r1/s?refs=processAppUsage&project=packages)**() //App级

**[processMiscUsage](http://androidxref.com/4.0.3_r1/s?refs=processMiscUsage&project=packages)**() //Device级

硬件单位时间耗电数据

[PowerProfile](http://androidxref.com/4.0.3_r1/s?defs=PowerProfile&project=packages) **[mPowerProfile](http://androidxref.com/4.0.3_r1/s?refs=mPowerProfile&project=packages)**; 加载com.android.internal.R.xml.power\_profile.xml文件

（国美手机：IUV-MJ-GM01-016）配置文件路径：

[platform/frameworks/base/core/res/res/xml/power\_profile.xml](https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/xml/power_profile.xml)

设备制造商必须提供组件的电源配置文件，该配置文件定义了组件的电流消耗值以及该组件在一段时间内大概消耗的电量

该文件参考官方文档：

<https://source.android.com/devices/tech/power/#power-values>

Profile常量数据集合

（国美手机：IUV-MJ-GM01-016）：

|  |  |  |
| --- | --- | --- |
| Name | Desc | Value |
| screen.on | 屏幕最低亮度 | 200mA |
| screen.full | 屏幕最高亮度 | 300mA |
| bluetooth.active | Bluetooth data transfer | 10mA |
| bluetooth.on | Bluetooth on & connectable, but not connected | 0.1mA |
| wifi.on | Wifi打开，未接受发送数据 | 3mA |
| wifi.active | WIFI data transfer | ~200mA |
| wifi.scan | WIFI network scanning | 100mA |
| dsp.audio | 当通过 DSP 进行音频解码/编码时消耗的额外电量 | 10mA |
| dsp.video | 当通过 DSP 进行视频解码时消耗的额外电量 | 50mA |
| camera.flashlight | Avg. power for camera flash | 160mA |
| camera.avg | Avg. power use of camera in standard usecases | 550mA |
| Mobile | | |
| radio.active | 蜂窝无线电发送/接收信号时消耗的额外电量 | 200mA |
| radio.scanning |  | 10mA |
| gps.on | GPS 获取信号时消耗的额外电量 | 50mA |
| radio.on | 在系统分页调度下，不同的无线信号强度下消耗的电流 | 2mA,1mA |
| CPU | | |
| cpu.speeds.cluster0 |  | 400000MHz CPU speed |
| cpu.idle | 当 CPU（和 SoC）处于系统挂起状态时，系统消耗的总电量 | 0.1mA |
| cpu.active | CPU 以不同速度运行时消耗的额外电量 | 100mA |
| cpu.active.cluster0 | CPU集群1 | 100mA |
|  | | |
| battery.capacity | 电池能力（以毫安时为单位） | 1000mAh |
| Wifi related values | | |
| wifi.controller.idle | Idle Receive current for wifi radio in mA. 0 by default | 0 |
| wifi.controller.rx | Rx current for wifi radio in mA. 0 by default | 0 |
| wifi.controller.tx | Tx current for wifi radio in mA. 0 by default | 0 |
| wifi.controller.voltage | Operating volatage for wifi radio in mV. 0 by default | 0 |
| wifi.batchedscan | Wifi批量扫描 | 1-8/hr ，9-64/hr ，65-512/hr ，513-4,096/hr ，4097-/hr |

App耗电量统计

private void processAppUsage(SparseArray<UserHandle> asUsers) {

final boolean forAllUsers = (asUsers.get(UserHandle.USER\_ALL) != null);

mStatsPeriod = mTypeBatteryRealtimeUs;

BatterySipper osSipper = null;

final SparseArray<? extends Uid> uidStats = mStats.getUidStats();

final int NU = uidStats.size();

498 for (int iu = 0; iu < NU; iu++) {

final Uid u = uidStats.valueAt(iu);

final BatterySipper app = new BatterySipper(BatterySipper.DrainType.APP, u, 0);

501 mCpuPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

502 mWakelockPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

503 mMobileRadioPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

504 mWifiPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

505 mBluetoothPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

506 mSensorPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

507 mCameraPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

508 mFlashlightPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

final double totalPower = app.sumPower();

if (DEBUG && totalPower != 0) {

Log.d(TAG, String.format("UID %d: total power=%s", u.getUid(),

makemAh(totalPower)));

}

// Add the app to the list if it is consuming power.

517 if (totalPower != 0 || u.getUid() == 0) {

518 //

519 // Add the app to the app list, WiFi, Bluetooth, etc, or into "Other Users" list.

520 //

521 final int uid = app.getUid();

522 final int userId = UserHandle.getUserId(uid);

523 if (uid == Process.WIFI\_UID) {

524 mWifiSippers.add(app);

525 } else if (uid == Process.BLUETOOTH\_UID) {

526 mBluetoothSippers.add(app);

527 } else if (!forAllUsers && asUsers.get(userId) == null

528 && UserHandle.getAppId(uid) >= Process.FIRST\_APPLICATION\_UID) {

529 // We are told to just report this user's apps as one large entry.

530 List<BatterySipper> list = mUserSippers.get(userId);

531 if (list == null) {

532 list = new ArrayList<>();

533 mUserSippers.put(userId, list);

534 }

535 list.add(app);

536 } else {

537 mUsageList.add(app);

538 }

539

540 if (uid == 0) {

541 osSipper = app;

542 }

543 }

544 }

545 if (osSipper != null) {

// The device has probably been awake for longer than the screen on

// time and application wake lock time would account for. Assign

// this remainder to the OS, if possible.

mWakelockPowerCalculator.calculateRemaining(osSipper, mStats, mRawRealtimeUs,

mRawUptimeUs, mStatsType);

osSipper.sumPower();

}

554 }

方法解析：

498：遍历所有APP，根据app的UID来标志每个App

501~508：分别计算每个APPCpu,WakeLock,MobileRadio,Wifi,BlueTooth,Sensor,Camera,FlashLight耗电量

517~544：将计算完成的App信息保存在BatterySipper对象中，最终方法mUseageList集合中

545~554：校正osApp的电量

501~508方法解析：

1.mCpuPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[CpuPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=CpuPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/CpuPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

33 long rawUptimeUs, int statsType) {

//计算CPU使用时间，cpuTimeMs 保存信息

35 app.cpuTimeMs = (u.getUserCpuTimeUs(statsType) + u.getSystemCpuTimeUs(statsType)) / 1000;

36

37 // Aggregate total time spent on each cluster.遍历所有CPU集群，得到使用时间的最大值

38 long totalTime = 0;

39 final int numClusters = mProfile.getNumCpuClusters();

40 for (int cluster = 0; cluster < numClusters; cluster++) {

41 final int speedsForCluster = mProfile.getNumSpeedStepsInCpuCluster(cluster);

42 for (int speed = 0; speed < speedsForCluster; speed++) {

43 totalTime += u.getTimeAtCpuSpeed(cluster, speed, statsType);

44 }

45 }

46 totalTime = Math.max(totalTime, 1);

47 //计算CPU耗电量 毫安/毫秒

48 double cpuPowerMaMs = 0;

49 for (int cluster = 0; cluster < numClusters; cluster++) {

50 final int speedsForCluster = mProfile.getNumSpeedStepsInCpuCluster(cluster);

51 for (int speed = 0; speed < speedsForCluster; speed++) {

52 final double ratio = (double) u.getTimeAtCpuSpeed(cluster, speed, statsType) /

53 totalTime;

54 final double cpuSpeedStepPower = ratio \* app.cpuTimeMs \*

55 mProfile.getAveragePowerForCpu(cluster, speed);

56 if (DEBUG && ratio != 0) {

57 Log.d(TAG, "UID " + u.getUid() + ": CPU cluster #" + cluster + " step #"

58 + speed + " ratio=" + BatteryStatsHelper.makemAh(ratio) + " power="

59 + BatteryStatsHelper.makemAh(cpuSpeedStepPower / (60 \* 60 \* 1000)));

60 }

61 cpuPowerMaMs += cpuSpeedStepPower;

62 }

63 }

//计算CPU耗电量，cpuPowerMah保存信息

64 app.cpuPowerMah = cpuPowerMaMs / (60 \* 60 \* 1000);

65

66 if (DEBUG && (app.cpuTimeMs != 0 || app.cpuPowerMah != 0)) {

67 Log.d(TAG, "UID " + u.getUid() + ": CPU time=" + app.cpuTimeMs + " ms power="

68 + BatteryStatsHelper.makemAh(app.cpuPowerMah));

69 }

70

71 // Keep track of the package with highest drain.

72 double highestDrain = 0;

73

74 app.cpuFgTimeMs = 0;

75 final ArrayMap<String, ? extends BatteryStats.Uid.Proc> processStats = u.getProcessStats();

76 final int processStatsCount = processStats.size();

77 for (int i = 0; i < processStatsCount; i++) {

78 final BatteryStats.Uid.Proc ps = processStats.valueAt(i);

79 final String processName = processStats.keyAt(i);

//计算CPU前台时间，cpuFgTimeMs 保存信息

80 app.cpuFgTimeMs += ps.getForegroundTime(statsType);

81

82 final long costValue = ps.getUserTime(statsType) + ps.getSystemTime(statsType)

83 + ps.getForegroundTime(statsType);

84

85 // Each App can have multiple packages and with multiple running processes.

86 // Keep track of the package who's process has the highest drain.

87 if (app.packageWithHighestDrain == null ||

88 app.packageWithHighestDrain.startsWith("\*")) {

89 highestDrain = costValue;

//得到出最高耗能进程，packageWithHighestDrain 保存信息

90 app.packageWithHighestDrain = processName;

91 } else if (highestDrain < costValue && !processName.startsWith("\*")) {

92 highestDrain = costValue;

//得到出最高耗能进程，packageWithHighestDrain 保存信息

93 app.packageWithHighestDrain = processName;

94 }

95 }

96

97 // Ensure that the CPU times make sense.

//矫正CPU时间与前台时间

98 if (app.cpuFgTimeMs > app.cpuTimeMs) {

99 if (DEBUG && app.cpuFgTimeMs > app.cpuTimeMs + 10000) {

100 Log.d(TAG, "WARNING! Cputime is more than 10 seconds behind Foreground time");

101 }

102

103 // Statistics may not have been gathered yet.

104 app.cpuTimeMs = app.cpuFgTimeMs;

105 }

106 }

方法影响值

CpuTimeMs: Cpu使用时间

CpuPowerMah: Cpu耗电值

CpuFgTimeMs: Cpu前台时间

packageWithHighestDrain: App高耗进程名(至少一个)

2.mWakelockPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[WakelockPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=WakelockPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/WakelockPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

34 long rawUptimeUs, int statsType) {

35 long wakeLockTimeUs = 0;

36 final ArrayMap<String, ? extends BatteryStats.Uid.Wakelock> wakelockStats =

37 u.getWakelockStats();

38 final int wakelockStatsCount = wakelockStats.size();

//得到每个wakelock的使用时间，单位us

39 for (int i = 0; i < wakelockStatsCount; i++) {

40 final BatteryStats.Uid.Wakelock wakelock = wakelockStats.valueAt(i);

41

42 // Only care about partial wake locks since full wake locks

43 // are canceled when the user turns the screen off.

44 BatteryStats.Timer timer = wakelock.getWakeTime(BatteryStats.WAKE\_TYPE\_PARTIAL);

45 if (timer != null) {

46 wakeLockTimeUs += timer.getTotalTimeLocked(rawRealtimeUs, statsType);

47 }

48 }

//计算wakeLock时间，wakeLockTimeMs保存信息

49 app.wakeLockTimeMs = wakeLockTimeUs / 1000; // convert to millis

50 mTotalAppWakelockTimeMs += app.wakeLockTimeMs;

51

52 // Add cost of holding a wake lock.

//计算wakeLock耗电量，wakeLockPowerMah 保存信息

53 app.wakeLockPowerMah = (app.wakeLockTimeMs \* mPowerWakelock) / (1000\*60\*60);

54 if (DEBUG && app.wakeLockPowerMah != 0) {

55 Log.d(TAG, "UID " + u.getUid() + ": wake " + app.wakeLockTimeMs

56 + " power=" + BatteryStatsHelper.makemAh(app.wakeLockPowerMah));

57 }

58 }

59

方法影响值

wakeLockTimeMs: wakeLock时间

wakeLockPowerMah : wakeLock耗电量

3.mMobileRadioPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[MobileRadioPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=MobileRadioPowerCalculator&project=frameworks)***

See<http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/MobileRadioPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

63 long rawUptimeUs, int statsType) {

64 // Add cost of mobile traffic.移动通信的消耗量

//tx是发送（transport），rx是接收(receive)

//mobileRxPackets移动通信接收的数据包

65 app.mobileRxPackets = u.getNetworkActivityPackets(BatteryStats.NETWORK\_MOBILE\_RX\_DATA,

66 statsType);

//mobileTxPackets 移动通信发送的数据包

67 app.mobileTxPackets = u.getNetworkActivityPackets(BatteryStats.NETWORK\_MOBILE\_TX\_DATA,

68 statsType);

//mobileActive 移动无线电活跃时间

69 app.mobileActive = u.getMobileRadioActiveTime(statsType) / 1000;

//mobileActiveCount 移动无线电数量

70 app.mobileActiveCount = u.getMobileRadioActiveCount(statsType);

//mobileRxBytes 移动无线电接收数据量

71 app.mobileRxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_MOBILE\_RX\_DATA,

72 statsType);

//mobileTxBytes 移动无线电发送数据量

73 app.mobileTxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_MOBILE\_TX\_DATA,

74 statsType);

75

76 if (app.mobileActive > 0) {

77 // We are tracking when the radio is up, so can use the active time to

78 // determine power use.

79 mTotalAppMobileActiveMs += app.mobileActive;

//计算无线电消耗电量，mobileRadioPowerMah记录值

80 app.mobileRadioPowerMah = (app.mobileActive \* mPowerRadioOn) / (1000\*60\*60);

81 } else {

82 // We are not tracking when the radio is up, so must approximate power use

83 // based on the number of packets.

//计算无线电消耗电量，mobileRadioPowerMah记录值

84 app.mobileRadioPowerMah = (app.mobileRxPackets + app.mobileTxPackets)

85 \* getMobilePowerPerPacket(rawRealtimeUs, statsType);

86 }

87 if (DEBUG && app.mobileRadioPowerMah != 0) {

88 Log.d(TAG, "UID " + u.getUid() + ": mobile packets "

89 + (app.mobileRxPackets + app.mobileTxPackets)

90 + " active time " + app.mobileActive

91 + " power=" + BatteryStatsHelper.makemAh(app.mobileRadioPowerMah));

92 }

93 }

方法影响值

mobileRxPackets移动通信接收的数据包

mobileTxPackets 移动通信发送的数据包

mobileActive 移动无线电活跃时间

mobileActiveCount 移动无线电数量

mobileRxBytes 移动无线电接收数据量

mobileTxBytes 移动无线电发送数据量

mobileRadioPowerMah 无线电消耗电量

4.mWifiPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

由于Profile.xml WIFI controller 值 = 0 ，所以取WifiPowerCalculator

实现类：***[WifiPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=WifiPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/WifiPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

42 long rawUptimeUs, int statsType) {

43 final BatteryStats.ControllerActivityCounter counter = u.getWifiControllerActivity();

44 if (counter == null) {

45 return;

46 }

47

48 final long idleTime = counter.getIdleTimeCounter().getCountLocked(statsType);

49 final long txTime = counter.getTxTimeCounters()[0].getCountLocked(statsType);

50 final long rxTime = counter.getRxTimeCounter().getCountLocked(statsType);

//wifiRunningTimeMs WIFI打开下运行时间

51 app.wifiRunningTimeMs = idleTime + rxTime + txTime;

52 mTotalAppRunningTime += app.wifiRunningTimeMs;

53

//wifiPowerMah WIFI使用耗电量(空闲时间，传递数据时间)

54 app.wifiPowerMah =

55 ((idleTime \* mIdleCurrentMa) + (txTime \* mTxCurrentMa) + (rxTime \* mRxCurrentMa))

56 / (1000\*60\*60);

57 mTotalAppPowerDrain += app.wifiPowerMah;

58

//wifiRxPackets WIFI使用接收数据包数量

59 app.wifiRxPackets = u.getNetworkActivityPackets(BatteryStats.NETWORK\_WIFI\_RX\_DATA,

60 statsType);

//wifiTxPackets WIFI使用发送数据包数量

61 app.wifiTxPackets = u.getNetworkActivityPackets(BatteryStats.NETWORK\_WIFI\_TX\_DATA,

62 statsType);

//wifiRxBytes WIFI使用接收数据量

63 app.wifiRxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_WIFI\_RX\_DATA,

64 statsType);

//wifiTxPackets WIFI使用发送数据量

65 app.wifiTxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_WIFI\_TX\_DATA,

66 statsType);

67

68 if (DEBUG && app.wifiPowerMah != 0) {

69 Log.d(TAG, "UID " + u.getUid() + ": idle=" + idleTime + "ms rx=" + rxTime + "ms tx=" +

70 txTime + "ms power=" + BatteryStatsHelper.makemAh(app.wifiPowerMah));

71 }

72 }

方法影响值

wifiRunningTimeMs WIFI打开下运行时间

wifiPowerMah WIFI使用耗电量(空闲时间，传递数据时间)

wifiRxPackets WIFI使用接收数据包数量

wifiTxPackets WIFI使用发送数据包数量

wifiRxBytes WIFI使用接收数据量

wifiTxPackets WIFI使用发送数据量

5.mBluetoothPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[BluetoothPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=BluetoothPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/BluetoothPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

38 long rawUptimeUs, int statsType) {

39

40 final BatteryStats.ControllerActivityCounter counter = u.getBluetoothControllerActivity();

41 if (counter == null) {

42 return;

43 }

44

45 final long idleTimeMs = counter.getIdleTimeCounter().getCountLocked(statsType);

46 final long rxTimeMs = counter.getRxTimeCounter().getCountLocked(statsType);

47 final long txTimeMs = counter.getTxTimeCounters()[0].getCountLocked(statsType);

48 final long totalTimeMs = idleTimeMs + txTimeMs + rxTimeMs;

49 double powerMah = counter.getPowerCounter().getCountLocked(statsType)

50 / (double)(1000\*60\*60);

51

52 if (powerMah == 0) {

53 powerMah = ((idleTimeMs \* mIdleMa) + (rxTimeMs \* mRxMa) + (txTimeMs \* mTxMa))

54 / (1000\*60\*60);

55 }

56

//bluetoothPowerMah 蓝牙耗电量

57 app.bluetoothPowerMah = powerMah;

//bluetoothRunningTimeMs 使用蓝牙时间

58 app.bluetoothRunningTimeMs = totalTimeMs;

//btRxBytes 接收到的数据

59 app.btRxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_BT\_RX\_DATA, statsType);

//btTxBytes 发送的数据

60 app.btTxBytes = u.getNetworkActivityBytes(BatteryStats.NETWORK\_BT\_TX\_DATA, statsType);

61

62 mAppTotalPowerMah += powerMah;

63 mAppTotalTimeMs += totalTimeMs;

64 }

方法影响值

bluetoothPowerMah 蓝牙耗电量

bluetoothRunningTimeMs 使用蓝牙时间

btRxBytes 接收到的数据

btTxBytes 发送的数据

6.mSensorPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[SensorPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=SensorPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/SensorPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

36 long rawUptimeUs, int statsType) {

37 // Process Sensor usage

38 final SparseArray<? extends BatteryStats.Uid.Sensor> sensorStats = u.getSensorStats();

39 final int NSE = sensorStats.size();

40 for (int ise = 0; ise < NSE; ise++) {

41 final BatteryStats.Uid.Sensor sensor = sensorStats.valueAt(ise);

42 final int sensorHandle = sensorStats.keyAt(ise);

43 final BatteryStats.Timer timer = sensor.getSensorTime();

44 final long sensorTime = timer.getTotalTimeLocked(rawRealtimeUs, statsType) / 1000;

45 switch (sensorHandle) {

46 case BatteryStats.Uid.Sensor.GPS:

//gpsTimeMs GPS使用时间

47 app.gpsTimeMs = sensorTime;

//gpsPowerMah Gps耗电量

48 app.gpsPowerMah = (app.gpsTimeMs \* mGpsPowerOn) / (1000\*60\*60);

49 break;

50 default:

51 final int sensorsCount = mSensors.size();

52 for (int i = 0; i < sensorsCount; i++) {

53 final Sensor s = mSensors.get(i);

54 if (s.getHandle() == sensorHandle) {

//sensorPowerMah 传感器耗电量(不含GPS)

55 app.sensorPowerMah += (sensorTime \* s.getPower()) / (1000\*60\*60);

56 break;

57 }

58 }

59 break;

60 }

61 }

62 }

方法影响值

gpsTimeMs GPS使用时间

gpsPowerMah Gps耗电量

sensorPowerMah 传感器耗电量(不含GPS)

7.mCameraPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[CameraPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=CameraPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/CameraPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

34 long rawUptimeUs, int statsType) {

35

36 // Calculate camera power usage. Right now, this is a (very) rough estimate based on the

37 // average power usage for a typical camera application.

38 final BatteryStats.Timer timer = u.getCameraTurnedOnTimer();

39 if (timer != null) {

40 final long totalTime = timer.getTotalTimeLocked(rawRealtimeUs, statsType) / 1000;

//cameraTimeMs 使用camera时间，默认0

41 app.cameraTimeMs = totalTime;

//cameraPowerMah 使用camera耗电量，默认0

42 app.cameraPowerMah = (totalTime \* mCameraPowerOnAvg) / (1000\*60\*60);

43 } else {

44 app.cameraTimeMs = 0;

45 app.cameraPowerMah = 0;

46 }

47 }

方法影响值

cameraTimeMs 使用camera时间，默认0

cameraPowerMah 使用camera耗电量，默认0

8.mFlashlightPowerCalculator.calculateApp(app, u, mRawRealtimeUs, mRawUptimeUs, mStatsType);

实现类：***[FlashlightPowerCalculator](http://androidxref.com/7.0.0_r1/s?refs=FlashlightPowerCalculator&project=frameworks)***

See <http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/FlashlightPowerCalculator.java>

public void calculateApp(BatterySipper app, BatteryStats.Uid u, long rawRealtimeUs,

32 long rawUptimeUs, int statsType) {

33

34 // Calculate flashlight power usage. Right now, this is based on the average power draw

35 // of the flash unit when kept on over a short period of time.

36 final BatteryStats.Timer timer = u.getFlashlightTurnedOnTimer();

37 if (timer != null) {

38 final long totalTime = timer.getTotalTimeLocked(rawRealtimeUs, statsType) / 1000;

//flashlightTimeMs 闪光灯使用时间，默认0

39 app.flashlightTimeMs = totalTime;

//flashlightPowerMah 闪光灯耗电量，默认0

40 app.flashlightPowerMah = (totalTime \* mFlashlightPowerOnAvg) / (1000\*60\*60);

41 } else {

42 app.flashlightTimeMs = 0;

43 app.flashlightPowerMah = 0;

44 }

45 }

方法影响值

flashlightTimeMs 闪光灯使用时间，默认0

flashlightPowerMah 闪光灯耗电量，默认0

共测量耗电值(应用级)

|  |  |  |
| --- | --- | --- |
| totalPowerMah | 总耗电量 | UNIT |
| CPU | | |
| CpuTimeMs | Cpu使用时间 | ms |
| CpuPowerMah | Cpu耗电值 | ms |
| CpuFgTimeMs | Cpu前台时间 | ms |
| packageWithHighestDrain | App高耗进程名(至少一个) | ms |
| wakeLockTimeMs | wakeLock时间 | byte |
| wakeLockPowerMah | wakeLock耗电量 |  |
| MOBILE | | |
| mobileRxPackets | 移动通信接收的数据包 |  |
| mobileTxPackets | 移动通信发送的数据包 |  |
| mobileActive | 移动无线电活跃时间 |  |
| mobileActiveCount | 移动无线电数量 |  |
| mobileRxBytes | 移动无线电接收数据量 |  |
| mobileTxBytes | 移动无线电发送数据量 |  |
| mobileRadioPowerMah | 无线电消耗电量 |  |
| WIFI | | |
| wifiRunningTimeMs WIFI | 打开下运行时间 |  |
| wifiPowerMah WIFI | 使用耗电量(空闲时间，传递数据时间) |  |
| wifiRxPackets WIFI | 使用接收数据包数量 |  |
| wifiTxPackets WIFI | 使用发送数据包数量 |  |
| wifiRxBytes WIFI | 使用接收数据量 |  |
| wifiTxPackets WIFI | 使用发送数据量 |  |
| wifiRunningTimeMs | WIFI打开下运行时间 |  |
| wifiPowerMah | WIFI耗电量 |  |
| BLUETOOTH | | |
| bluetoothPowerMah | 蓝牙耗电量 |  |
| bluetoothRunningTimeMs | 使用蓝牙时间 |  |
| btRxBytes | 接收到的数据 |  |
| btTxBytes | 发送的数据 |  |
| bluetoothPowerMah; | 蓝牙耗电量 |  |
| GPS | | |
| gpsTimeMs | GPS使用时间 |  |
| gpsPowerMah | Gps耗电量 |  |
| sensorPowerMah | 传感器耗电量(不含GPS) |  |
| CAMERA | | |
| cameraTimeMs | 使用camera时间，默认0 |  |
| cameraPowerMah | 使用camera耗电量，默认0 |  |
| FLASHLIGHT | | |
| flashlightTimeMs | 闪光灯使用时间，默认0 |  |
| flashlightPowerMah | 闪光灯耗电量，默认0 |  |

硬件耗电量统计

processMiscUsage()分析：

//将设备耗电记录在List<BatterySipper> mUsageList中，同APP

private void processMiscUsage() {

final int which = mStatsType;

long uSecTime = SystemClock.elapsedRealtime() \* 1000;

final long uSecNow = mStats.computeBatteryRealtime(uSecTime, which);

final long timeSinceUnplugged = uSecNow;

if (DEBUG) {

Log.i(TAG, "Uptime since last unplugged = " + (timeSinceUnplugged / 1000));

}

//通话耗电量PowerProfile.POWER\_RADIO\_ACTIVE

addPhoneUsage(uSecNow);

//屏幕耗电量PowerProfile.POWER\_SCREEN\_ON

addScreenUsage(uSecNow);

//WIFI耗电量PowerProfile.POWER\_WIFI\_ON

addWiFiUsage(uSecNow);

//蓝牙耗电量PowerProfile.POWER\_BLUETOOTH\_ON

addBluetoothUsage(uSecNow);

//binder通信耗电量PowerProfile.POWER\_CPU\_IDLE

addIdleUsage(uSecNow); // Not including cellular idle power

// Don't compute radio usage if it's a wifi-only device

if (!com.android.settings.Utils.isWifiOnly(getActivity())) {

//PowerProfile.POWER\_RADIO\_ON

addRadioUsage(uSecNow);

}

}

private void addPhoneUsage() {

long phoneOnTimeMs = mStats.getPhoneOnTime(mRawRealtimeUs, mStatsType) / 1000;

double phoneOnPower = mPowerProfile.getAveragePower(PowerProfile.POWER\_RADIO\_ACTIVE)

\* phoneOnTimeMs / (60\*60\*1000);

if (phoneOnPower != 0) {

addEntry(BatterySipper.DrainType.PHONE, phoneOnTimeMs, phoneOnPower);

}

}

private void addScreenUsage() {

double power = 0;

long screenOnTimeMs = mStats.getScreenOnTime(mRawRealtimeUs, mStatsType) / 1000;

power += screenOnTimeMs \* mPowerProfile.getAveragePower(PowerProfile.POWER\_SCREEN\_ON);

final double screenFullPower =

mPowerProfile.getAveragePower(PowerProfile.POWER\_SCREEN\_FULL);

for (int i = 0; i < BatteryStats.NUM\_SCREEN\_BRIGHTNESS\_BINS; i++) {

double screenBinPower = screenFullPower \* (i + 0.5f)

/ BatteryStats.NUM\_SCREEN\_BRIGHTNESS\_BINS;

long brightnessTime = mStats.getScreenBrightnessTime(i, mRawRealtimeUs, mStatsType)

/ 1000;

double p = screenBinPower\*brightnessTime;

if (DEBUG && p != 0) {

Log.d(TAG, "Screen bin #" + i + ": time=" + brightnessTime

+ " power=" + makemAh(p / (60 \* 60 \* 1000)));

}

power += p;

}

power /= (60\*60\*1000); // To hours

if (power != 0) {

addEntry(BatterySipper.DrainType.SCREEN, screenOnTimeMs, power);

}

}

/\*\*

\* We do per-app blaming of WiFi activity. If energy info is reported from the controller,

\* then only the WiFi process gets blamed here since we normalize power calculations and

\* assign all the power drain to apps. If energy info is not reported, we attribute the

\* difference between total running time of WiFi for all apps and the actual running time

\* of WiFi to the WiFi subsystem.

\*/

private void addWiFiUsage() {

BatterySipper bs = new BatterySipper(DrainType.WIFI, null, 0);

mWifiPowerCalculator.calculateRemaining(bs, mStats, mRawRealtimeUs, mRawUptimeUs, mStatsType);

aggregateSippers(bs, mWifiSippers, "WIFI");

if (bs.totalPowerMah > 0) {

mUsageList.add(bs);

}

}

/\*\*

\* Bluetooth usage is not attributed to any apps yet, so the entire blame goes to the

\* Bluetooth Category.

\*/

private void addBluetoothUsage() {

BatterySipper bs = new BatterySipper(BatterySipper.DrainType.BLUETOOTH, null, 0);

mBluetoothPowerCalculator.calculateRemaining(bs, mStats, mRawRealtimeUs, mRawUptimeUs,

mStatsType);

aggregateSippers(bs, mBluetoothSippers, "Bluetooth");

if (bs.totalPowerMah > 0) {

mUsageList.add(bs);

}

}

private void addIdleUsage(long uSecNow) {

long idleTimeMs = (uSecNow - mStats.getScreenOnTime(uSecNow, mStatsType)) / 1000;

double idlePower = (idleTimeMs \* mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_IDLE))

/ 1000;

addEntry(getActivity().getString(R.string.power\_idle), DrainType.IDLE, idleTimeMs,

R.drawable.ic\_settings\_phone\_idle, idlePower);

}

/\*\*

\* Calculate the baseline power usage for the device when it is in suspend and idle.

\* The device is drawing POWER\_CPU\_IDLE power at its lowest power state.

\* The device is drawing POWER\_CPU\_IDLE + POWER\_CPU\_AWAKE power when a wakelock is held.

\*/

private void addIdleUsage() {

final double suspendPowerMaMs = (mTypeBatteryRealtimeUs / 1000) \*

mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_IDLE);

final double idlePowerMaMs = (mTypeBatteryUptimeUs / 1000) \*

mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_AWAKE);

final double totalPowerMah = (suspendPowerMaMs + idlePowerMaMs) / (60 \* 60 \* 1000);

if (DEBUG && totalPowerMah != 0) {

Log.d(TAG, "Suspend: time=" + (mTypeBatteryRealtimeUs / 1000)

+ " power=" + makemAh(suspendPowerMaMs / (60 \* 60 \* 1000)));

Log.d(TAG, "Idle: time=" + (mTypeBatteryUptimeUs / 1000)

+ " power=" + makemAh(idlePowerMaMs / (60 \* 60 \* 1000)));

}

if (totalPowerMah != 0) {

addEntry(BatterySipper.DrainType.IDLE, mTypeBatteryRealtimeUs / 1000, totalPowerMah);

}

}

private BatterySipper addEntry(DrainType drainType, long time, double power) {

BatterySipper bs = new BatterySipper(drainType, null, 0);

bs.usagePowerMah = power;

bs.usageTimeMs = time;

bs.sumPower();

mUsageList.add(bs);

return bs;

}

测量耗电值(device级)

|  |  |
| --- | --- |
| usagePowerMah | 总耗电量 |
| usageTimeMs | 使用时间 |

耗电信息汇总

|  |  |  |
| --- | --- | --- |
| totalPowerMah | 总耗电量 | mA |
| CPU | | |
| CpuTimeMs | Cpu使用时间 |  |
| CpuPowerMah | Cpu耗电值 |  |
| CpuFgTimeMs | Cpu前台时间 |  |
| packageWithHighestDrain | App高耗进程名(至少一个) |  |
| wakeLockTimeMs | wakeLock时间 | 200mA |
| wakeLockPowerMah | wakeLock耗电量 |  |
| MOBILE | | |
| mobileRxPackets | 移动通信接收的数据包 |  |
| mobileTxPackets | 移动通信发送的数据包 |  |
| mobileActive | 移动无线电活跃时间 |  |
| mobileActiveCount | 移动无线电数量 |  |
| mobileRxBytes | 移动无线电接收数据量 |  |
| mobileTxBytes | 移动无线电发送数据量 |  |
| mobileRadioPowerMah | 无线电消耗电量 |  |
| WIFI | | |
| wifiRunningTimeMs | WIFI打开下运行时间 |  |
| wifiPowerMah | WIFI使用耗电量(空闲时间，传递数据时间) |  |
| wifiRxPackets | WIFI使用接收数据包数量 |  |
| wifiTxPackets | WIFI使用发送数据包数量 |  |
| wifiRxBytes | WIFI使用接收数据量 |  |
| wifiTxPackets | WIFI使用发送数据量 |  |
| BLUETOOTH | | |
| bluetoothPowerMah | 蓝牙耗电量 |  |
| bluetoothRunningTimeMs | 使用蓝牙时间 |  |
| btRxBytes | 接收到的数据 |  |
| btTxBytes | 发送的数据 |  |
| GPS | | |
| gpsTimeMs | GPS使用时间 |  |
| gpsPowerMah | Gps耗电量 |  |
| sensorPowerMah | 传感器耗电量(不含GPS) |  |
| CAMERA | | |
| cameraTimeMs | 使用camera时间，默认0 |  |
| cameraPowerMah | 使用camera耗电量，默认0 |  |
| FLASHLIGHT | | |
| flashlightTimeMs | 闪光灯使用时间，默认0 |  |
| flashlightPowerMah | 闪光灯耗电量，默认0 |  |

对于Device应用：

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | DESC | CALCULATOR | UNIT |
| usagePowerMah | 总耗电量 |  |  |
| usageTimeMs | 使用时间 |  |  |
| [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages) | 类型 | [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[PHONE](http://androidxref.com/4.0.3_r1/s?defs=PHONE&project=packages)  [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[SCREEN](http://androidxref.com/4.0.3_r1/s?defs=SCREEN&project=packages)  [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[CELL](http://androidxref.com/4.0.3_r1/s?defs=CELL&project=packages)  [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[WIFI](http://androidxref.com/4.0.3_r1/s?defs=WIFI&project=packages)  [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[IDLE](http://androidxref.com/4.0.3_r1/s?defs=IDLE&project=packages)  [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[BLUETOOTH](http://androidxref.com/4.0.3_r1/s?defs=BLUETOOTH&project=packages) | ENUM |
| 其他属性 | 不可用 | 不可用 | 不可用 |

参考 ***[BatterySipper](http://androidxref.com/7.0.0_r1/s?refs=BatterySipper&project=frameworks)***

<http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/BatterySipper.java>

Q1解决方案

Framework层

提供systemService

1.1抽取***[BatteryStatsHelper](http://androidxref.com/7.0.0_r1/s?refs=BatteryStatsHelper&project=frameworks).java***

[List](http://androidxref.com/4.0.3_r1/s?defs=List&project=packages)<[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)> **[mUsageList](http://androidxref.com/4.0.3_r1/s?refs=mUsageList&project=packages)**

[List](http://androidxref.com/4.0.3_r1/s?defs=List&project=packages)<[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)> **[mWifiSippers](http://androidxref.com/4.0.3_r1/s?refs=mWifiSippers&project=packages)**

[List](http://androidxref.com/4.0.3_r1/s?defs=List&project=packages)<[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)> **[mBluetoothSippers](http://androidxref.com/4.0.3_r1/s?refs=mBluetoothSippers&project=packages)**

**[mTotalPower](http://androidxref.com/4.0.3_r1/s?refs=mTotalPower&project=packages)**

1.2 提供***[BatterySipper](http://androidxref.com/7.0.0_r1/s?refs=BatterySipper&project=frameworks).java***类，或者自定义类实现parcelable，提取所有成员变量

App层

根据得到的**[mUsageList](http://androidxref.com/4.0.3_r1/s?refs=mUsageList&project=packages)**的数据集，可以得到每个APP耗电量的详细信息，APP调用每个组件所用的时间，总耗电量，单个组件的耗电量，然后进行一系列的逻辑计算，判断APP的耗电量属于什么标准，是正常耗电还是高耗电或者异常耗电

Q2解决方案

目前FrameWork已经分析完成，下面走到可视化层APP，研究耗电量怎样才能满足正常值，高值和异常高值

这部分有几个问题

单个APP每个部件耗电占比怎么计算，判定一个APP耗电级数是多少需要满足什么样的算法？

需要确定App耗电量的标准值，异常值是多少？

首先针对第一个问题(单个APP每个部件耗电占比怎么计算)

分析APP级各部分参数，如下

各组件耗电排行

|  |  |  |
| --- | --- | --- |
| DESC | VALUE | Profile |
| 摄像头打开 | 550mA | Camera.active |
| 屏幕高亮 | 300mA | Screen.full |
| 屏幕常亮 | 200mA | Screen.on |
| 数据(tcpreceived+tcpsend) | 200mA | Radio.active+Wifi.active |
| 摄像头闪光 | 160mA | Camera.flashLight |
| cpu运行(active)wakeLock | 100mA | cpu.active |
| Wifi扫描 | 100mA | Wifi.scan |
| GPS运行(active) | 50mA | GPS.on |
| 移动数据扫描 | 10mA | Radio.scan |
| 蓝牙传递数据 | 10mA | Bluetooth.active |
| Wifi Running(打开状态下) | 3mA | Wifi.on |

可见使用摄像头,屏幕,数据请求时耗电是最大的，那么算法可以讲这一点作为高标准评分。

算法模拟

1总耗电评分

如果总电量占比高，应用可能高耗电 BatterySipper.totalPowerMah / BatteryStatsHelper.mTotalPower 达到25%，第一点总高耗电满足(表现形)

2 CPU评分

如果CpuTimeMs - CpuFgTimeMs算出CPU后台时间 / CpuFgTimeMs总时间占比90%，表示APP后台处理频率高，则第二点可能存在高耗电(潜在形)

|  |  |  |
| --- | --- | --- |
| CPU | | |
| CpuTimeMs | Cpu使用时间 |  |
| CpuPowerMah | Cpu耗电值 |  |
| CpuFgTimeMs | Cpu前台时间 |  |

3 Camera项评分

使用Camera最为耗电

|  |  |  |
| --- | --- | --- |
| CPU | | |
| CpuTimeMs | Cpu使用时间 |  |
| CpuPowerMah | Cpu耗电值 |  |
| CpuFgTimeMs | Cpu前台时间 |  |
| CAMERA | | |
| cameraTimeMs | 使用camera时间，默认0 |  |
| cameraPowerMah | 使用camera耗电量，默认0 |  |
| FLASHLIGHT | | |
| flashlightTimeMs | 闪光灯使用时间，默认0 |  |
| flashlightPowerMah | 闪光灯耗电量，默认0 |  |

CpuTimeMs = 600s

cameraTimeMs = 180s

CpuFgTimeMs = 200s

3.1计算使用camera的时间在cpu时间中的占比，(cameraTimeMs +flashlightTimeMs )/ CpuTimeMs一般达到30%属于高耗电

计算使用camera在app使用总电量占比(cameraPowerMah +flashlightPowerMah ）/BatterySipper.totalPowerMah 达到50%属于高耗电(表现形)

第三点高耗电满足

4使用数据传输

|  |  |  |
| --- | --- | --- |
| MOBILE | | |
| mobileRxPackets | 移动通信接收的数据包 |  |
| mobileTxPackets | 移动通信发送的数据包 |  |
| mobileActive | 移动无线电活跃时间 |  |
| mobileActiveCount | 移动无线电数量 |  |
| mobileRxBytes | 移动无线电接收数据量 |  |
| mobileTxBytes | 移动无线电发送数据量 |  |
| mobileRadioPowerMah | 无线电消耗电量 |  |
| WIFI | | |
| wifiRunningTimeMs | WIFI打开下运行时间 |  |
| wifiPowerMah | WIFI使用耗电量(空闲时间，传递数据时间) |  |
| wifiRxPackets | WIFI使用接收数据包数量 |  |
| wifiTxPackets | WIFI使用发送数据包数量 |  |
| wifiRxBytes | WIFI使用接收数据量 |  |
| wifiTxPackets | WIFI使用发送数据量 |  |
| BLUETOOTH | | |
| bluetoothPowerMah | 蓝牙耗电量 |  |
| bluetoothRunningTimeMs | 使用蓝牙时间 |  |
| btRxBytes | 接收到的数据 |  |
| btTxBytes | 发送的数据 |  |

4.1 计算数据传输占比 ，由于数据传出的数据量和耗电量成正比，我们可以将数据传输耗电量与总数据耗电量比较，得出比例

mobileRadioPowerMah + wifiPowerMah +bluetoothPowerMah / BatterySipper.totalPowerMah达到30%属于高耗电(潜在形)

第四点可能存在高耗电

Q3解决方案

数字：Q2中条件

|  |  |
| --- | --- |
| 1 | 可能高耗电(表现型) |
| 3 | 可能高耗电(表现型) |
| 1+3 | 高耗电(表现型) |
|  | |
| 1+2 | 必定高耗电 |
| 1+4 | 必定高耗电 |
| 3+2 | 必定高耗电 |
| 3+4 | 必定高耗电 |
|  | |
| 2+4 | 必定高耗电(后台网络请求过多，高耗电) |
|  | |
| 1+2+3 | 异常耗电 |
| 2+3+4 | 异常耗电 |
| 1+2+3+4 | 异常耗电 |

Reference

<https://source.android.com/devices/tech/power/#power-values> [电源参数文件1]

<https://source.android.com/devices/tech/power/values> [电源参数文件2]

<http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageSummary.java> [framework层系统统计应用耗电详细1]

<http://androidxref.com/7.0.0_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageBase.java> [framework层系统统计应用耗电详细2]

<http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/BatteryStatsHelper.java>

[framework层系统统计应用耗电详细3]

<http://androidxref.com/7.0.0_r1/xref/frameworks/base/core/java/com/android/internal/os/BatterySipper.java>

[framework层系统统计应用耗电详细4]

Attachment

power\_profile.xml

（国美手机：IUV-MJ-GM01-016）配置文件路径：

[platform/frameworks/base/core/res/res/xml/power\_profile.xml](https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/xml/power_profile.xml)

<?xml version="1.0" encoding="utf-8"?>

<!--

\*\*

\*\* Copyright 2009, The Android Open Source Project

\*\*

\*\* Licensed under the Apache License, Version 2.0 (the "License")

\*\* you may not use this file except in compliance with the License.

\*\* You may obtain a copy of the License at

\*\*

\*\* http://www.apache.org/licenses/LICENSE-2.0

\*\*

\*\* Unless required by applicable law or agreed to in writing, software

\*\* distributed under the License is distributed on an "AS IS" BASIS,

\*\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\*\* See the License for the specific language governing permissions and

\*\* limitations under the License.

\*/

-->

<device name="Android">

<!-- Most values are the incremental current used by a feature,

in mA (measured at nominal voltage).

The default values are deliberately incorrect dummy values.

OEM's must measure and provide actual values before

shipping a device.

Example real-world values are given in comments, but they

are totally dependent on the platform and can vary

significantly, so should be measured on the shipping platform

with a power meter. -->

<item name="none">0</item>

<item name="screen.on">200</item> <!-- ~200mA -->

<item name="screen.full">300</item> <!-- ~300mA -->

<item name="bluetooth.active">10</item> <!-- Bluetooth data transfer, ~10mA -->

<item name="bluetooth.on">0.1</item> <!-- Bluetooth on & connectable, but not connected, ~0.1mA -->

<item name="wifi.on">3</item> <!-- ~3mA -->

<item name="wifi.active">200</item> <!-- WIFI data transfer, ~200mA -->

<item name="wifi.scan">100</item> <!-- WIFI network scanning, ~100mA -->

<item name="dsp.audio">100</item> <!-- ~10mA -->

<item name="dsp.video">50</item> <!-- ~50mA -->

<item name="camera.flashlight">0.1</item> <!-- Avg. power for camera flash, ~160mA -->

<item name="camera.avg">0.1</item> <!-- Avg. power use of camera in standard usecases, ~550mA -->

<!-- Radio related values. For modems without energy reporting support in firmware, use

radio.active, radio.scanning, and radio.on. -->

<item name="radio.active">200</item> <!-- ~200mA -->

<item name="radio.scanning">10</item> <!-- cellular radio scanning for signal, ~10mA -->

<item name="gps.on">50</item> <!-- ~50mA -->

<!-- Current consumed by the radio at different signal strengths, when paging -->

<array name="radio.on"> <!-- Strength 0 to BINS-1 -->

<value>2</value> <!-- ~2mA -->

<value>1</value> <!-- ~1mA -->

</array>

<!-- Radio related values. For modems WITH energy reporting support in firmware, use

modem.controller.idle, modem.controller.tx, modem.controller.rx, modem.controller.voltage.

-->

<item name="modem.controller.idle">0</item>

<item name="modem.controller.rx">0</item>

<item name="modem.controller.tx">0</item>

<item name="modem.controller.voltage">0</item>

<!-- A list of heterogeneous CPU clusters, where the value for each cluster represents the

number of CPU cores for that cluster.

Ex:

<array name="cpu.clusters.cores">

<value>4</value> // cluster 0 has cpu0, cpu1, cpu2, cpu3

<value>2</value> // cluster 1 has cpu4, cpu5

</array> -->

<array name="cpu.clusters.cores">

<value>1</value> <!-- cluster 0 has cpu0 -->

</array>

<!-- Different CPU speeds for cluster 0 as reported in

/sys/devices/system/cpu/cpu0/cpufreq/stats/time\_in\_state.

There must be one of these for each cluster, labeled:

cpu.speeds.cluster0, cpu.speeds.cluster1, etc... -->

<array name="cpu.speeds.cluster0">

<value>400000</value> <!-- 400 MHz CPU speed -->

</array>

<!-- Current when CPU is idle -->

<item name="cpu.idle">100</item>

<!-- Current at each CPU speed, as per 'cpu.speeds' -->

<array name="cpu.active">

<value>100</value> <!-- ~100mA -->

</array>

<!-- Current at each CPU speed for cluster 0, as per 'cpu.speeds.cluster0'.

Like cpu.speeds.cluster0, there must be one of these present for

each heterogeneous CPU cluster. -->

<array name="cpu.active.cluster0">

<value>0.1</value> <!-- ~100mA -->

</array>

<!-- Current when CPU is idle -->

<item name="cpu.idle">0.1</item>

<!-- This is the battery capacity in mAh (measured at nominal voltage) -->

<item name="battery.capacity">1000</item>

<!-- Wifi related values. -->

<!-- Idle Receive current for wifi radio in mA. 0 by default-->

<item name="wifi.controller.idle">0</item>

<!-- Rx current for wifi radio in mA. 0 by default-->

<item name="wifi.controller.rx">0</item>

<!-- Tx current for wifi radio in mA. 0 by default-->

<item name="wifi.controller.tx">0</item>

<!-- Current at each of the wifi Tx levels in mA. The number of tx levels varies per device

and is available only of wifi chipsets which support the tx level reporting. Use

wifi.tx for other chipsets. none by default -->

<array name="wifi.controller.tx\_levels"> <!-- mA -->

</array>

<!-- Operating volatage for wifi radio in mV. 0 by default-->

<item name="wifi.controller.voltage">0</item>

<array name="wifi.batchedscan"> <!-- mA -->

<value>.2</value> <!-- 1-8/hr -->

<value>2</value> <!-- 9-64/hr -->

<value>20</value> <!-- 65-512/hr -->

<value>200</value> <!-- 513-4,096/hr -->

<value>500</value> <!-- 4097-/hr -->

</array>

</device>