# APP耗电量解决方案

问题：

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，算法？统计出的耗电信息时间单位是时还是分钟？

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：

<http://androidxref.com/4.0.3_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageSummary.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占用电流数据实现类

[IBatteryStats](http://androidxref.com/4.0.3_r1/s?defs=IBatteryStats&project=packages) **[mBatteryInfo](http://androidxref.com/4.0.3_r1/s?refs=mBatteryInfo&project=packages)**; 用于获取系统服务‘batteryinfo’ service [***[BatteryStatsService](http://androidxref.com/4.0.3_r1/s?refs=BatteryStatsService&project=frameworks)***]

[BatteryStatsImpl](http://androidxref.com/4.0.3_r1/s?defs=BatteryStatsImpl&project=packages) **[mStats](http://androidxref.com/4.0.3_r1/s?refs=mStats&project=packages)**;由**[mBatteryInfo](http://androidxref.com/4.0.3_r1/s?refs=mBatteryInfo&project=packages)**得到

***[BatteryStatsService](http://androidxref.com/4.0.3_r1/s?refs=BatteryStatsService&project=frameworks)*** 得到/data/system/batterystats.bin文件，该文件记录不同app使用电量

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：processAppUsage()  分析：

private void processAppUsage() {

//得到传感器服务

SensorManager sensorManager = (SensorManager)getActivity().getSystemService(

Context.SENSOR\_SERVICE);

//上一次拔掉设备后 ~ 至今 的App耗电量统计 mStatsType = BatteryStats.STATS\_SINCE\_CHARGED;

final int which = mStatsType;

final int speedSteps = mPowerProfile.getNumSpeedSteps();

final double[] powerCpuNormal = new double[speedSteps];

final long[] cpuSpeedStepTimes = new long[speedSteps];

//得到CPU在不同速率下面的耗电量

for (int p = 0; p < speedSteps; p++) {

powerCpuNormal[p] = mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_ACTIVE, p);

}

final double averageCostPerByte = getAverageDataCost();

long uSecTime = mStats.computeBatteryRealtime(SystemClock.elapsedRealtime() \* 1000, which);

long appWakelockTime = 0;

BatterySipper osApp = null;

mStatsPeriod = uSecTime;

//Uid 代表着一个APP，由App签名和sharedUserId决定

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

final int NU = uidStats.size();

//for循环得到各个App信息

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

Uid u = uidStats.valueAt(iu);

//初始化power值，默认0mA

double power = 0;

double highestDrain = 0;

String packageWithHighestDrain = null;

//mUsageList.add(new AppUsage(u.getUid(), new double[] {power}));

//得到App下面每个进程耗电数据

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

//CPU运行时间

long cpuTime = 0;

//前台运行时间

long cpuFgTime = 0;

//设备唤醒时间

long wakelockTime = 0;

//gps占用时间

long gpsTime = 0;

if (processStats.size() > 0) {

// Process CPU time

for (Map.Entry<String, ? extends BatteryStats.Uid.Proc> ent

: processStats.entrySet()) {

if (DEBUG) Log.i(TAG, "Process name = " + ent.getKey()); //Log进程名

Uid.Proc ps = ent.getValue();

//使用时间+内核运行时间

final long userTime = ps.getUserTime(which);

final long systemTime = ps.getSystemTime(which);

//前台运行时间

final long foregroundTime = ps.getForegroundTime(which);

cpuFgTime += foregroundTime \* 10; // convert to millis 转为毫秒？

//该进程使用CPU运行时间

final long tmpCpuTime = (userTime + systemTime) \* 10; // convert to millis

cpuTime += tmpCpuTime;

//进程所用电量

int totalTimeAtSpeeds = 0;

// 获取使用总时间

for (int step = 0; step < speedSteps; step++) {

cpuSpeedStepTimes[step] = ps.getTimeAtCpuSpeedStep(step, which);

totalTimeAtSpeeds += cpuSpeedStepTimes[step];

}

if (totalTimeAtSpeeds == 0) totalTimeAtSpeeds = 1;

// Then compute the ratio of time spent at each speed

double processPower = 0;

//CPU在不同速率下面的耗电量

for (int step = 0; step < speedSteps; step++) {

double ratio = (double) cpuSpeedStepTimes[step] / totalTimeAtSpeeds;

processPower += ratio \* tmpCpuTime \* powerCpuNormal[step];

}

//processPower表示CPU耗电量

power += processPower;

power /= 1000;

if (packageWithHighestDrain == null

|| packageWithHighestDrain.startsWith("\*")) {

highestDrain = processPower;

packageWithHighestDrain = ent.getKey();

} else if (highestDrain < processPower

&& !ent.getKey().startsWith("\*")) {

highestDrain = processPower;

packageWithHighestDrain = ent.getKey();

}

}

if (DEBUG) Log.i(TAG, "Max drain of " + highestDrain

+ " by " + packageWithHighestDrain);

}

//前台时间与CPU所用时间

if (cpuFgTime > cpuTime) {

if (DEBUG && cpuFgTime > cpuTime + 10000) {

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

}

cpuTime = cpuFgTime; // Statistics may not have been gathered yet.

}

//得到每个唤醒锁的时间

// Process wake lock usage

Map<String, ? extends BatteryStats.Uid.Wakelock> wakelockStats = u.getWakelockStats();

for (Map.Entry<String, ? extends BatteryStats.Uid.Wakelock> wakelockEntry

: wakelockStats.entrySet()) {

Uid.Wakelock wakelock = wakelockEntry.getValue();

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

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

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

if (timer != null) {

wakelockTime += timer.getTotalTimeLocked(uSecTime, which);

}

}

wakelockTime /= 1000; // convert to millis

appWakelockTime += wakelockTime;

//唤醒屏幕的耗电量

// Add cost of holding a wake lock

power += (wakelockTime

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_AWAKE)) / 1000;

//数据流量的电量使用

// Add cost of data traffic

long tcpBytesReceived = u.getTcpBytesReceived(mStatsType);

long tcpBytesSent = u.getTcpBytesSent(mStatsType);

//averageCostPerByte表示WIFI与移动数据平均单位时间耗电量

power += (tcpBytesReceived+tcpBytesSent) \* averageCostPerByte;

//使用wifi的耗电量

// Add cost of keeping WIFI running.

long wifiRunningTimeMs = u.getWifiRunningTime(uSecTime, which) / 1000;

mAppWifiRunning += wifiRunningTimeMs;

power += (wifiRunningTimeMs

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_WIFI\_ON)) / 1000;

//传感器的耗电量

// Process Sensor usage

Map<Integer, ? extends BatteryStats.Uid.Sensor> sensorStats = u.getSensorStats();

for (Map.Entry<Integer, ? extends BatteryStats.Uid.Sensor> sensorEntry

: sensorStats.entrySet()) {

Uid.Sensor sensor = sensorEntry.getValue();

int sensorType = sensor.getHandle();

BatteryStats.Timer timer = sensor.getSensorTime();

long sensorTime = timer.getTotalTimeLocked(uSecTime, which) / 1000;

double multiplier = 0;

switch (sensorType) {

case Uid.Sensor.GPS:

multiplier = mPowerProfile.getAveragePower(PowerProfile.POWER\_GPS\_ON);

gpsTime = sensorTime;

break;

default:

android.hardware.Sensor sensorData =

sensorManager.getDefaultSensor(sensorType);

if (sensorData != null) {

multiplier = sensorData.getPower();

if (DEBUG) {

Log.i(TAG, "Got sensor " + sensorData.getName() + " with power = "

+ multiplier);

}

}

}

power += (multiplier \* sensorTime) / 1000;

}

//总电量total

if (DEBUG) Log.i(TAG, "UID " + u.getUid() + ": power=" + power);

//将耗电信息放入App信息中(BatterySipper.java)

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

if (power != 0 || u.getUid() == 0) {

BatterySipper app = new BatterySipper(getActivity(), mRequestQueue, mHandler,

packageWithHighestDrain, DrainType.APP, 0, u,

new double[] {power});

app.cpuTime = cpuTime;

app.gpsTime = gpsTime;

app.wifiRunningTime = wifiRunningTimeMs;

app.cpuFgTime = cpuFgTime;

app.wakeLockTime = wakelockTime;

app.tcpBytesReceived = tcpBytesReceived;

app.tcpBytesSent = tcpBytesSent;

if (u.getUid() == Process.WIFI\_UID) {

mWifiSippers.add(app);

} else if (u.getUid() == Process.BLUETOOTH\_GID) {

mBluetoothSippers.add(app);

} else {

mUsageList.add(app);

}

if (u.getUid() == 0) {

osApp = app;

}

}

//wifi总用电量

if (u.getUid() == Process.WIFI\_UID) {

mWifiPower += power;

//蓝牙总用电量

} else if (u.getUid() == Process.BLUETOOTH\_GID) {

mBluetoothPower += power;

} else {

if (power > mMaxPower) mMaxPower = power; //总用电量

mTotalPower += power;

}

if (DEBUG) Log.i(TAG, "Added power = " + power);

}

//修改OS\_APP的电量参数 UID = 0

// 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.

if (osApp != null) {

long wakeTimeMillis = mStats.computeBatteryUptime(

SystemClock.uptimeMillis() \* 1000, which) / 1000;

wakeTimeMillis -= appWakelockTime - (mStats.getScreenOnTime(

SystemClock.elapsedRealtime(), which) / 1000);

if (wakeTimeMillis > 0) {

double power = (wakeTimeMillis

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_CPU\_AWAKE)) / 1000;

osApp.wakeLockTime += wakeTimeMillis;

osApp.value += power;

osApp.values[0] += power;

if (osApp.value > mMaxPower) mMaxPower = osApp.value;

mTotalPower += power;

}

}

}

共测量耗电值如下(应用级)：

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | DESC | CALCULATOR | UNIT |
| Power | 总耗电量 |  |  |
| Cputime | 使用cpu时间 | 每个进程使用cpu时间 + 每个进程在内核运行时间 | ms |
| Gpstime | 使用gps时间 | GPS传感器打开情况下所用时间 | ms |
| WifiRunningTime | 使用WIF时间 | 使用wifi时间(在wifi打开情况下) | ms |
| CpuFGTime | 应用在前台使用CPU时间 | 每个进程在前台运行时间总和 | ms |
| WakelockTime | 应用唤醒设备时间 | 每次唤醒wakelock使用时间总和 | ms |
| TcpByteReceived | tcp接收到的数据 |  | byte |
| TcpByteSend | tcp发送的数据量 |  | byte |

power应用总用电量计算值：

Powersingle = ms[每个部件使用时间] \* PowerProfile.getAveragePower(Type)[每个部件单位时间耗电量] / 1000

PowerTotal = CpuPower + SensorPower[GPS..]+ WifiRunningPower + WakelockPower + TcpPower[数据传递平均耗电量 wifi + mobile]

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

http://androidxref.com/4.0.3\_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/BatterySipper.java

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 uSecNow) {

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

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

\* phoneOnTimeMs / 1000;

addEntry(getActivity().getString(R.string.power\_phone), DrainType.PHONE, phoneOnTimeMs,

R.drawable.ic\_settings\_voice\_calls, phoneOnPower);

}

private void addScreenUsage(long uSecNow) {

double power = 0;

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

power += screenBinPower \* brightnessTime;

if (DEBUG) {

Log.i(TAG, "Screen bin power = " + (int) screenBinPower + ", time = "

+ brightnessTime);

}

}

power /= 1000; // To seconds

addEntry(getActivity().getString(R.string.power\_screen), DrainType.SCREEN, screenOnTimeMs,

R.drawable.ic\_settings\_display, power);

}

private void addWiFiUsage(long uSecNow) {

long onTimeMs = mStats.getWifiOnTime(uSecNow, mStatsType) / 1000;

long runningTimeMs = mStats.getGlobalWifiRunningTime(uSecNow, mStatsType) / 1000;

if (DEBUG) Log.i(TAG, "WIFI runningTime=" + runningTimeMs

+ " app runningTime=" + mAppWifiRunning);

runningTimeMs -= mAppWifiRunning;

if (runningTimeMs < 0) runningTimeMs = 0;

double wifiPower = (onTimeMs \* 0 /\* TODO \*/

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_WIFI\_ON)

+ runningTimeMs \* mPowerProfile.getAveragePower(PowerProfile.POWER\_WIFI\_ON)) / 1000;

if (DEBUG) Log.i(TAG, "WIFI power=" + wifiPower + " from procs=" + mWifiPower);

BatterySipper bs = addEntry(getActivity().getString(R.string.power\_wifi), DrainType.WIFI,

runningTimeMs, R.drawable.ic\_settings\_wifi, wifiPower + mWifiPower);

aggregateSippers(bs, mWifiSippers, "WIFI");

}

private void addBluetoothUsage(long uSecNow) {

long btOnTimeMs = mStats.getBluetoothOnTime(uSecNow, mStatsType) / 1000;

double btPower = btOnTimeMs \* mPowerProfile.getAveragePower(PowerProfile.POWER\_BLUETOOTH\_ON)

/ 1000;

int btPingCount = mStats.getBluetoothPingCount();

btPower += (btPingCount

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_BLUETOOTH\_AT\_CMD)) / 1000;

BatterySipper bs = addEntry(getActivity().getString(R.string.power\_bluetooth),

DrainType.BLUETOOTH, btOnTimeMs, R.drawable.ic\_settings\_bluetooth,

btPower + mBluetoothPower);

aggregateSippers(bs, mBluetoothSippers, "Bluetooth");

}

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);

}

private void addRadioUsage(long uSecNow) {

double power = 0;

final int BINS = SignalStrength.NUM\_SIGNAL\_STRENGTH\_BINS;

long signalTimeMs = 0;

for (int i = 0; i < BINS; i++) {

long strengthTimeMs = mStats.getPhoneSignalStrengthTime(i, uSecNow, mStatsType) / 1000;

power += strengthTimeMs / 1000

\* mPowerProfile.getAveragePower(PowerProfile.POWER\_RADIO\_ON, i);

signalTimeMs += strengthTimeMs;

}

long scanningTimeMs = mStats.getPhoneSignalScanningTime(uSecNow, mStatsType) / 1000;

power += scanningTimeMs / 1000 \* mPowerProfile.getAveragePower(

PowerProfile.POWER\_RADIO\_SCANNING);

BatterySipper bs =

addEntry(getActivity().getString(R.string.power\_cell), DrainType.CELL,

signalTimeMs, R.drawable.ic\_settings\_cell\_standby, power);

if (signalTimeMs != 0) {

bs.noCoveragePercent = mStats.getPhoneSignalStrengthTime(0, uSecNow, mStatsType)

/ 1000 \* 100.0 / signalTimeMs;

}

}

private BatterySipper addEntry(String label, DrainType drainType, long time, int iconId,

double power) {

if (power > mMaxPower) mMaxPower = power;

mTotalPower += power;

BatterySipper bs = new BatterySipper(getActivity(), mRequestQueue, mHandler,

label, drainType, iconId, null, new double[] {power});

bs.usageTime = time;

bs.iconId = iconId;

mUsageList.add(bs);

return bs;

}

测量耗电值如下(device级)

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | DESC | CALCULATOR | UNIT |
| mTotalPower | 总用电量 |  | mA |
| PowerProfile.POWER\_SCREEN\_ON | 屏幕耗电量 | addScreenUsage() | mA |
| PowerProfile.POWER\_RADIO\_ACTIVE | 通话耗电量 | addPhoneUsage() | mA |
| PowerProfile.POWER\_WIFI\_ON | WIFI耗电量 | addWiFiUsage() | mA |
| PowerProfile.POWER\_BLUETOOTH\_ON | 蓝牙耗电量 | addBluetoothUsage() | mA |
| PowerProfile.POWER\_CPU\_IDLE) | CPU空闲耗电量 | addIdleUsage() | mA |
| PowerProfile.POWER\_RADIO\_ON | 不是仅支持WIFI设备下  无线耗电量 | addRadioUsage() | mA |

mTotalPower = 屏幕耗电量 + 通话耗电量 + WIFI耗电量 +蓝牙耗电量 + CPU空闲耗电量 + 无线耗电量

耗电信息汇总:

[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)>(); //耗电集合

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

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

**[mBluetoothPower](http://androidxref.com/4.0.3_r1/s?refs=mBluetoothPower&project=packages)** 蓝牙耗电量

**[mUsageList](http://androidxref.com/4.0.3_r1/s?refs=mUsageList&project=packages)** 包括了系统级所有应用耗电详情，[BatterySipper](http://androidxref.com/4.0.3_r1/s?defs=BatterySipper&project=packages)耗电实体类详情

对于普通应用：

可用属性：

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | DESC | CALCULATOR | UNIT |
| Power | 总耗电量 |  |  |
| Cputime | 使用cpu时间 | 每个进程使用cpu时间 + 每个进程在内核运行时间 | ms |
| Gpstime | 使用gps时间 | GPS传感器打开情况下所用时间 | ms |
| WifiRunningTime | 使用WIF时间 | 使用wifi时间(在wifi打开情况下) | ms |
| CpuFGTime | 应用在前台使用CPU时间 | 每个进程在前台运行时间总和 | ms |
| WakelockTime | 应用唤醒设备时间 | 每次唤醒wakelock使用时间总和 | ms |
| TcpByteReceived | tcp接收到的数据 |  | byte |
| TcpByteSend | tcp发送的数据量 |  | byte |
| [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages) | 类型 = App 包括osApp | [DrainType](http://androidxref.com/4.0.3_r1/s?defs=DrainType&project=packages).[APP](http://androidxref.com/4.0.3_r1/s?defs=APP&project=packages) | ENUM |

对于Device应用：

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | DESC | CALCULATOR | UNIT |
| Power | 总耗电量 |  |  |
| useageTime | 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 |

See ***[BatterySipper](http://androidxref.com/4.0.3_r1/s?refs=BatterySipper&project=packages).java*** <http://androidxref.com/4.0.3_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/BatterySipper.java>

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

<http://androidxref.com/4.0.3_r1/xref/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageDetail.java#58>

问题一解决方案：

Framework层：

提供systemService

1.1抽取***[PowerUsageSummary](http://androidxref.com/4.0.3_r1/s?refs=PowerUsageSummary&project=packages)*** .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)**

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

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

以上6个成员变量

1.2 提供***[BatterySipper](http://androidxref.com/4.0.3_r1/s?refs=BatterySipper&project=packages).java***类，或者自定义类实现parcelable，提取

name,packageName,icon drainType,Power,cpuTime,gpsTime,wifiRunningTime,cpuFgTime,wakeLocakTime,tcpByteReceived,tcpBytesSend，useageTime等成员变量

源码未暴露但需要添加的新成员变量：

单位时间每部件的耗电量，用于更加详细的计算出每个部件的耗电量，目前仅仅得到部件占用时间在APP层不能计算出其具体耗电量CpuPower , SensorPower[GPS..], WifiRunningPower , WakelockPower , TcpPower[数据传递平均耗电量 wifi + mobile]

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

各组件耗电排行

|  |  |
| --- | --- |
| 数据(tcpreceived+tcpsend) | 200mA |
| cpu运行(active) | 100mA |
| wakeLock(屏幕) | 100mA |
| GPS运行(active) | 50mA |
| Wifi Running(打开状态下) | 3mA |
| Cpu挂起 | 0.1mA |

可见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，应用程序对网络访问的越多，那么就会越耗电，还有Cpu以及wakeLock耗电也比较大

Q2研究方案：

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

|  |  |  |  |
| --- | --- | --- | --- |
| Power | 单个APP总耗电量 |  |  |
| Cputime | 使用cpu时间 | 每个进程使用cpu时间 + 每个进程在内核运行时间 | ms |
| Gpstime | 使用gps时间 | GPS传感器打开情况下所用时间 | ms |
| WifiRunningTime | 使用WIF时间 | 使用wifi时间(在wifi打开情况下) | ms |
| CpuFGTime | 应用在前台使用CPU时间 | 每个进程在前台运行时间总和 | ms |
| WakelockTime | 应用唤醒设备时间 | 每次唤醒wakelock使用时间总和 | ms |
| TcpByteReceived | tcp接收到的数据 |  | byte |
| TcpByteSend | tcp发送的数据量 |  | byte |
| TcpPower | 数据传递平均耗电量 wifi + mobile |  | /200mA |
| CpuPower | 使用CPU电量 |  | /100mA |
| WakelockPower | 应用唤醒设备电量 |  | /100mA |
| SensorPower | 使用GPS等Sensor电量 |  | /50mA |
| WifiRunningPower | 使用WIFI电量 |  | /3mA |

APP应用层：

Android 6.0 ->Doze模式

RTC唤醒周期太频繁；唤醒周期低于30s  --- Alarmmanager

应用持锁时间过长 -- wakelock

后台应用CPU负载高 ,单个应用在后台运行时CPU占有率超过20%，且持续10分钟，则作为异常耗电应用提示用户。

ApplicationErrorReport.BatteryInfo