# 架构

ANDROID 系统网络连接和管理服务由四个系统服务ConnectivityService、NetworkPolicyManagerService、NetworkManagementService、NetworkStatsService共同配合完成网络连接和管理功能，四个服务只有ConnectivityService、NetworkPolicyManagerService两个服务通过NetworkPolicyManager、ConnectivityManager两个客户端对象对应用程序提供对外SDK接口，而NetworkManagementService、NetworkStatsService没有对外提供SDK接口,但设置应用程序可以通过获取系统服务接口使用NetworkManagementService、NetworkStatsService服务.

四个服务之间的关系类图如下：



ConnectivityService提供数据连接管理服务，NetworkPolicyManagerService提供网络策略管理服务，NetworkStatsService提供网络传输数据统计服务，NetworkManagementService提供对物理网络接口的管理服务，connectivityService也包括VPN、Tethering对象提供虚拟连接及共享连接管理。

ConnectivityService、NetworkPolicyManagerService、NetworkStatsService三个服务都通过INetworkManagementService接口跨进程访问NetworkManagementService服务，实现与网络接口的交互及信息读取。

NetworkStatsService、NetworkPolicyManagerService两个服务还通过IConnectivityManager接口与connectivityService服务通讯，从connectivityService读取网络连接的信息及打开数据连接的策略控制。

ConnectivityService服务也通过INetworkPolicyManager接口调用NetworkPolicyManagerService的API，读取网络限额信息，登记监听对象。connectivityService服务通过NetworkPolicyManagerService服务的registerListener函数向NetworkPolicyManagerService服务注册一个INetworkPolicyListener.Stub监听桩对象。 NetworkPolicyManagerService通过该监听对象的远程代理接口向ConnectivityService服务传送规则变化通知。

另外ConnectivityService服务的Tethering、VPN对象及NetworkPolicyManagerService、NetworkStatsService服务的内部NetworkAlertObserver类型的对象都直接或间接派生自INetworkManagementEventObserver.Stub，且四个对象都登记为NetworkManagementService的监听对象，NetworkManagementService服务通过INetworkManagementEventObserver接口向这些对象传送网络接口事件通知。

NetworkPolicyManagerService维护网络使用策略，策略可以从一个策略文件读取（策略文件保存在系统目录下的netpolicy.xml文件中）。也可以通过NetworkPolicyManager对外提供的设置策略接口（setNetworkPolicies及setUidPolicy）进行设置，NetworkPolicyManagerService能够根据这些设置或从策略文件中读取的策略控制网络连接。另外NetworkPolicyManagerService还具有动态调节网络连接限额及动态设置网络连接的功能，动态调节网络连接限额机制是通过INetworkStatsService访问NetworkStatsService服务获得上面设置或读取的策略匹配的网络连接类型的传输统计信息（NetworkPolicyManagerService采用NetworkTemplate进行网络连接类型的匹配），并根据这些信息生成有效的规则，并提交给ConnectivityService服务，并调用NetworkManagementService的setInterfaceQuota函数对网络连接的带宽限额进行控制。

动态设置网络连接规则的机制是NetworkPolicyManagerService服务通过检测系统发出的一些相关事件（在NetworkPolicyManagerService的启动systemReady函数中注册），包括ActivityManager服务中IProcessObserver的onForegroundActivitiesChanged及onProcessDied回调事件，NetworkManager服务中INetworkManagementEventObserver的limitReached回调事件，以及ACTION\_SCREEN\_ON 、CONNECTIVITY\_ACTION\_IMMEDIATE、ACTION\_PACKAGE\_ADDED、ACTION\_UID\_REMOVED、ACTION\_NETWORK\_STATS\_UPDATED、ACTION\_ALLOW\_BACKGROUND等INTENT事件，当这些事件发生时，根据事件不同对网络规则进行不同设置，如与应用程序相关的事件调用updateRulesForUidLocked函数对uid涉及的NetworkRule进行更新，其它事件通过updateNetworkEnabledLocked函数调用connectivityService的setPolicyDataEnable函数对特定网络连接类型的数据连接进行设置。

NetworkStatsService服务定期调用performPoll函数获得网络传输统计信息，performPoll函数通过调用NetworkManagementService服务的getNetworkStatsUidDetail、getNetworkStatsSummary及getNetworkStatsTethering函数从/proc/目录下的包含网络传输统计数据的文件中读取网络统计信息，并转换为NetworkStatsHistory数据结构，保存到以网络接口名称对应的NetworkIdentitySet类型和UID对应的UidStatsKey类型的变量为key的NetworkStatsService的三个HashMap变量中。然后根据performPoll传进来的参数标志信息（指示不同的PERSIST方法）调用writeNetworkDevStatsLocked、writeNetworkXtStatsLocked、writeUidStatsLocked函数把HashMap变量中的统计信息分别写入系统目录下的三个相应的BIN文件（netstats.bin、netstats\_xt.bin、netstats\_uid.bin）中。

NetworkManagementService的一个重要功能是与本地netd进程进行通讯，完成对网络物理接口的操作。NetworkManagementService通过NativeDaemonConnector与本地netd进程通过LocalSocket建立连接进行双向通讯，发送命令，读取事件和命令应答消息，对网络接口的实际操作由netd进程完成。NativeDaemonConnector对象是一个实现Runnable接口的对象，NativeDaemonConnector对象在NetworkManagementService创建的线程中运行。

NativeDaemonConnector通过实例化时从NetworkManagementService传进来的回调函数向NetworkManagementService传送从netd进程读取的事件，主要事件有接口增加、接口移出、接口状态变化、LINK状态变化等接口改变事件以及带宽控制事件。

# ConnectivityService

# netd

Netd（Network Daemon ），表示Network守护进程，类似的命名还有很多，例如 Vold（Volumn Deamon）---磁盘管理，Rild（Radio Interface Layer Deamon）--- 电话的基本数据功能……类似的还有好多，遍及Android各类服务，各个层次~

Netd负责跟一些涉及物理端口的网络操作相关的功能实现，例如带宽控制（Bandwidth），网络地址转换（NAT），个人局域网（pan），PPP链接，soft-ap，共享上网（Tether）等等……都是按照模块（.cpp+.h）组织在netd文件目录下的~

基本框架的四大部分：

(1)Linux Kernel 用于检测：network 相关的所有 event 事件。

(2)Netd 作为 Kernel 与：Framework 之间通信的桥梁。

(3)Framework 层操作：Netd,向 Netd 发送操作命令。

(4)UI 与 Framework：交互,用于用户进行网络的操控。

主要源码位置：

Netd:

/System/netd

/system/core/libsysutils/src

/system/core/include/sysutils

Framework:

/frameworks/base/services/java/com/android/server

**框架图**





NetworkmanagementService

此模块运行在SystemService中，负责Java层的实现机制，提供对上层的一些运行接口，当然，上层是通过一些抽象类实现进程间通讯进行访问的。

NetD

此模块是C++的Daemon，负责底层部分对于一些关键网络服务的管理。对上面Java服务提供接口，采用进程间通讯的方式。

Wpa\_supplicant

此模块是提供WIFI支持的模块，不做详细描述了。在Android中是一个关键的底层服务。

Dnsmasq

此服务实现了DHCP Server，用于辅助Hostapd，实现IP的管理。

Hosted

此服务实现了WIFI AP的关键服务，直接控制底层设备，此服务正常运行后，其他终端可以搜索到AP，并连接。

**从init.rc文件中可以看到，是在启动就开始运行的一个系统级的守护进程。而且同 Vold 基本并列~ 对于init.rc文件的意义请参考附录2**

****

## 如何开区一个softap

Soft AP代表通过软件实现Access Point的功能,

从功能角度来看，AP作为基站设备，起着连接其他无线设备到有线网的作用，相当于有线网络中的HUB与交换机。在日常工作和家庭中经常使用的无线路由器就是一个AP。一般情况下，它一端接着有线网络，另一端连接其他无线设备。

Station代表配备无线网络接口的设备，如手机、笔记本等。

虽然AP和Station是两个不同的设备，但实际上在Station中用软件也能实现AP拥有的功能，如桥接、路由等。在基本功能上，Soft AP与AP并没有太大的差别，只是Soft AP设备的接入能力和覆盖范围不如AP。

在Android系统中使用Soft AP功能还得借助另一个开源软件“hostapd”，这是一个运行在用户空间的用于AP和认证服务器的守护进程。它实现了IEEE 802.11相关的接入管理、IEEE 802.1X/WPA/WPA2/EAP 认证、RADIUS客户端、EAP服务器和RADIUS认证服务器。

以下是启动WIFI AP 的流程：



### 设置并开启 softap(便携式 WLAN 热点)

在 Setting 选项中进行设置,该设置功能的开启将涉及相关的UI 路径下相关的文件,packages /apps/Settings/src/com/android/settings/wifi 路径下,程序中的 WifiApEnabler. OnPreferenceChange 里, 设置 soft ap 在 wifiApDialog.onClick 里。

### 调用相应的处理函数,通过 socket 向 netd 下发命令。

对于 app 层的函数调用关系不做详细介绍,最终会调用到 Framework 层的 NetworkManagementService.startAccessPoint 函数。

frameworks\base\services\core\java\com\android\server

**[java]** [view plain](http://blog.csdn.net/xiaokeweng/article/details/8130218) [copy](http://blog.csdn.net/xiaokeweng/article/details/8130218)

1. //NetworkManagementService.java
2. @Override
3. **public** **void** startAccessPoint(
4. WifiConfiguration wifiConfig, String wlanIface, String softapIface) {
5. mContext.enforceCallingOrSelfPermission(CONNECTIVITY\_INTERNAL, TAG);
6. **try** {
7. wifiFirmwareReload(wlanIface, "AP");
8. **if** (wifiConfig == **null**) {
9. mConnector.execute("softap", "set", wlanIface, softapIface);
10. } **else** {
11. mConnector.execute("softap", "set", wlanIface, softapIface, wifiConfig.SSID,
12. getSecurityType(wifiConfig), wifiConfig.preSharedKey);
13. }
14. mConnector.execute("softap", "startap");
15. } **catch** (NativeDaemonConnectorException e) {
16. **throw** e.rethrowAsParcelableException();
17. }
18. }

### **netd处理,并将反馈给 Framework**

Netd 中 softap 控制的功能在/system/netd/SoftapController.{h,cpp}里, 具体的执行是通过调用网卡驱动的 ap 功能。NetworkManagementService 通过 NativeDaemonConnector 向下通过 socket 向下 softap 相关的字符串命令,NativeDaemonConnector 中维护着与 Netd 中 CommandListener 相关联的内部socket 线程。两者可以通过它相互通信。

**[cpp]** [view plain](http://blog.csdn.net/xiaokeweng/article/details/8130218) [copy](http://blog.csdn.net/xiaokeweng/article/details/8130218)

1. //CommandListener.cpp
2. int CommandListener::SoftapCmd::runCommand(SocketClient \*cli,
3. int argc, char \*\*argv) {
4. int rc = ResponseCode::SoftapStatusResult;
5. char \*retbuf = NULL;
6. if (sSoftapCtrl == NULL) {
7. cli->sendMsg(ResponseCode::ServiceStartFailed, "SoftAP is not available", false);
8. return -1;
9. }
10. if (argc < 2) {
11. cli->sendMsg(ResponseCode::CommandSyntaxError,
12. "Missing argument in a SoftAP command", false);
13. return 0;
14. }
15. if (!strcmp(argv[1], "startap")) {
16. rc = sSoftapCtrl->startSoftap();
17. } else if (!strcmp(argv[1], "stopap")) {
18. rc = sSoftapCtrl->stopSoftap();
19. } else if (!strcmp(argv[1], "fwreload")) {
20. rc = sSoftapCtrl->fwReloadSoftap(argc, argv);
21. } else if (!strcmp(argv[1], "status")) {
22. asprintf(&retbuf, "Softap service %s running",
23. (sSoftapCtrl->isSoftapStarted() ? "is" : "is not"));
24. cli->sendMsg(rc, retbuf, false);
25. free(retbuf);
26. return 0;
27. } else if (!strcmp(argv[1], "set")) {
28. **rc = sSoftapCtrl->setSoftap(argc, argv);**
29. } else {
30. cli->sendMsg(ResponseCode::CommandSyntaxError, "Unrecognized SoftAP command", false);
31. return 0;
32. }
33. if (rc >= 400 && rc < 600)
34. cli->sendMsg(rc, "SoftAP command has failed", false);
35. else
36. cli->sendMsg(rc, "Ok", false);
37. return 0;
38. }……

**sSoftapCtrl = new SoftapController();**

## SoftAP

### （2）SoftapCmd命令使用

和TetherCmd类似，开启[Android](https://www.2cto.com/kf/yidong/Android/)中手机的Soft AP功能将涉及大量Framework层中的操作，本节仅关注和Netd相关的三个步骤。

1）首先为Wi-Fi加载不同的固件（Firmware），这是通过SoftapController的fwReloadSoftap函数完成的，代码如下所示。

#### SoftapController->fwReloadSoftap

int SoftapController::fwReloadSoftap(int argc, char \*argv[])  
{  
    int ret, i = 0;  
    char \*iface;  
    char \*fwpath;

   ......// 参数检测  
    iface = argv[2];  
    if (strcmp(argv[3], "AP") == 0) {  
        fwpath = (char \*)wifi\_get\_fw\_path(WIFI\_GET\_FW\_PATH\_AP);  
    } else if (strcmp(argv[3], "P2P") == 0) {  
        fwpath = (char \*)wifi\_get\_fw\_path(WIFI\_GET\_FW\_PATH\_P2P);  
    } else {  
        fwpath = (char \*)wifi\_get\_fw\_path(WIFI\_GET\_FW\_PATH\_STA);  
    }  
    // 通过往/sys/module/wlan/parameters/fwpath文件中写入固件名  
    // 触发驱动去加载对应的固件  
    ret = wifi\_change\_fw\_path((const char \*)fwpath);  
    ......  
    return ret;  
}

上面这段代码表示在Android中，如果要让Wi-Fi无线设备扮演不同的角色，得为它们加载不同的固件（Firmware），具体说明如下。

WIFI\_GET\_FW\_PATH\_AP：代表Soft AP功能的固件，其对应的文件位置由WIFI\_DRIVER\_FW\_PATH\_AP宏表达。三星Tuna平台中，该文件位置为/vendor/firmware/fw\_bcmdhd\_apsta.bin。

WIFI\_GET\_FW\_PATH\_P2P：代表P2P功能的固件，其对应的文件位置由WIFI\_DRIVER\_FW\_PATH\_P2P宏表达。三星Tuna平台中，该文件位置为/vendor/firmware/ fw\_bcmdp2p.bin。

WIFI\_GET\_FW\_PATH\_STA：代表Station功能的固件，其对应的文件位置由WIFI\_DRIVER\_FW\_PATH\_STA宏表达。三星Tuna平台中，该文件位置为/vendor/firmware/fw\_bcmdhd.bin。

提示　三星Tuna平台对应的配置文件在Android 4.2[源码](https://www.2cto.com/ym/)根目录/device/samsung/tuna目录中。从上面的固件文件名来看，它用的Wi-Fi无线芯片是博通（Broadcom）公司生产的。通过加载不同固件的方式来启用无线芯片硬件的不同功能可能和Wi-Fi驱动及芯片的设计有关。

另外，根据审稿专家的反馈，在Android 4.2中，STA和P2P可同时运行（即所谓的共存模式），这样STA和P2P实际对应的固件相同，但可能文件名不同。而SoftAP的固件与STA/P2P就不一样了。

#### SoftapController.cpp::setSoftap

2）加载完指定的Wi-Fi固件后，下一步将对Soft AP功能进行一些配置，配置信息最终将写到一个配置文件。这部分功能由SoftapController的setSoftap函数完成，代码如下所示。

int SoftapController::setSoftap(int argc, char \*argv[]) {  
    char psk\_str[2\*SHA256\_DIGEST\_LENGTH+1];  
    int ret = 0, i = 0, fd;  
    char \*ssid, \*iface;

    ......// 参数检查

    iface = argv[2];

    char \*wbuf = NULL;  
    char \*fbuf = NULL;

    if (argc > 3) {  
        ssid = argv[3];  
    } else {  
        ssid = (char \*)"AndroidAP"; // SSID即接入点的名称  
    }

[asp](https://www.2cto.com/kf/web/asp/)rintf(&wbuf, "interface=%s\ndriver=nl80211\nctrl\_interface="  
            "/data/misc/wifi/hostapd\nssid=%s\nchannel=6\nieee80211n=1\n",  
            iface, ssid);

    if (argc > 4) { // 判断AP的[加密](https://www.2cto.com/article/jiami/)类型  
        if (!strcmp(argv[4], "wpa-psk")) {  
            generatePsk(ssid, argv[5], psk\_str);  
            [asp](https://www.2cto.com/kf/web/asp/)rintf(&fbuf, "%swpa=1\nwpa\_pairwise=TKIP CCMP\nwpa\_psk=%s\n",  
                      wbuf, psk\_str);  
        } else if (!strcmp(argv[4], "wpa2-psk")) {  
            generatePsk(ssid, argv[5], psk\_str);  
            asprintf(&fbuf, "%swpa=2\nrsn\_pairwise=CCMP\nwpa\_psk=%s\n",  
                      wbuf, psk\_str);  
        } else if (!strcmp(argv[4], "open")) {  
            asprintf(&fbuf, "%s", wbuf);  
        }  
    }  ......  
    // HOSTAPD\_CONF\_FILE指向/data/misc/wifi/hostapd.conf文件  
    fd = open(HOSTAPD\_CONF\_FILE, O\_CREAT | O\_TRUNC | O\_WRONLY, 0660);  
    ......  
    if (write(fd, fbuf, strlen(fbuf)) < 0) {  
        ALOGE("Cannot write to \"%s\": %s", HOSTAPD\_CONF\_FILE, strerror(errno));  
        ret = -1;  
    }  
    ......// 修改该文件的读写权限等  
    return ret;  
}

上面代码中涉及Wi-Fi技术的很多概念，将在后续章节统一介绍。从功能上来说，setSoftap函数无非就是把一些配置信息写到一个hostapd.conf文件中。可以通过一个例子文件来了解此文件的内容。

Android4.2/hardware/ti/wlan/mac80211/config目录中有一个hostapd.conf文件，其内容如下所示。  
[-->hostapd.conf]  
driver=nl80211   #指定Wi-Fi驱动的名称  
......#略去部分内容  
ssid=AndroidAP   #设置接入点名称为AndroidAP  
country\_code=US  
wep\_rekey\_period=0  
eap\_server=0  
own\_ip\_addr=127.0.0.1  
wpa\_group\_rekey=0  
wpa\_gmk\_rekey=0   #[加密](https://www.2cto.com/article/jiami/)方式等设置  
wpa\_ptk\_rekey=0  
interface=wlan1   #网络设备接口  
......#略去部分内容

由上边示例的hostapd.conf可知，当使用该配置文件后，其他Station搜索到由这台手机设置的Soft AP的名称将会是“AndroidAP”。

#### SoftapController的startap

3）最后，SoftapController的startap函数被调用，它将启动hostapd进程。重点关注hostapd启动的参数信息，如下所示。  
hostapd \  
-e /data/misc/wifi/entropy.bin \和Wi-Fi协议中的信息加密有关  
/data/misc/wifi/hostapd.conf \hostapd的配置文件

不同的wifi芯片需要不同的驱动支持。请参考external/wpa\_supplicant/README

1：例如通用的Broadcom wl.0 driver，它用于，Broadcom IEEE 802.11a/g cards，同时支持wifi热点和无线路由，即service端和client端。可以这样说，wifi热点和无线wifi是互斥的，两者不能共存。

关于Broadcom wl.0 driver，他不需要通过hostap的方式实现热点与driver的通信。而是直接在system/netd/SoftapController.cpp中与驱动通信。而system/netd/SoftapController.cpp与framework的通信则是通过socket。framework中NetworkManagementService.java中的startAccessPoint方法调用doCommand方法像SoftapController通过socket发送请求，并在NativeDaemonConnector.java中调用listenToSocket方法监听上报的事件，并向事件广播给app层。 之前在NetworkManagementService.java中的startAccessPoint中通过向下发送fwreload的请求来关闭wifi，打开wifi热点，但这样做会引起后期多次切换无线wifi和wifi热点的过程，打开无线wifi和wifi热点出错的Bug。之后改进的方法是在hardware/wifi/wifi.c中显式的load firmware。

2：另外madwifi driver 基于Atheros chip，这个Atheros在wifi.c中随处可见，但没用到。

3：ATMEL AT76C5XXx driver 用于USB和PCMCIA

4：其实移植android2.3。4的源码时，如果需要增加wifi热点功能，只需把softap的interface改成wl0.1就ok了。除非修改Bug，否则不需要大改，因为android2.2之后的版本本身就支持wifi热点。   
5：另外如果出现以下Bug：”保存添加的网络后重新开机，不显示添加的网络“或者“设备重启后WIFI不会自动连接上次保存好的ap”时，则需要考虑是否wpa\_supplicant.conf有可写的权限。可考虑修改init.rc。将service wpa\_supplicant /system/bin/logwrapper /system/bin/wpa\_supplicant -Dwext -ieth0 -c/system/etc/wifi/wpa\_supplicant.conf 改成   
service wpa\_supplicant /system/bin/logwrapper /system/bin/wpa\_supplicant -Dwext -ieth0 -c/data/misc/wifi/wpa\_supplicant.conf   
6：当wifi启用静态IP，并且设置的静态IP，网关，网络掩码都正确时，即使连接的ap本身不能上网，也不影响启用静态IP后能正常访问网络。

## 实战分析

### Softap热点原理分析

## 流量统计

在没有Root的情况下，Android应用流量统计在6.0之前一直没有太好的办法，官方虽然提供了TrafficStats，但其主要功能是设备启动以来流量的统计信息，和时间信息无法很好的配合。最近再看TrafficStats类时，发现说明中提到，为获取更具鲁棒性的网络历史数据，建议使用NetworkStatsManager。

本文首先简单对比下TrafficStats和NetworkStatsManager各自的限制和优缺点，然后详细说明NetworkStatsManager的用法，并给出主要代码。

### TrafficStats

Android API8提供了android.net.TrafficStats类。 通过此类能获取设备重启以来网络信息，部分函数如下所示：

1. **static** long  getMobileRxBytes()  //获取通过移动数据网络收到的字节总数static long  getMobileTxBytes()  //通过移动数据网发送的总字节数  static long  getTotalRxBytes()  //获取设备总的接收字节数 static long  getTotalTxBytes()  //获取设备总的发送字节数static long  getUidRxBytes(int uid)  //获取指定uid的接收字节数  static long  getUidTxBytes(int uid) //获取指定uid的发送字节数

通过文档及上述函数可以知道，TrafficStats能够获取设备的数据流量和总的网络流量消耗（一般情况下也就得到Wi-Fi下的流量信息）；可以查询uid对应的流量信息，而uid可以通过应用的包名查询到，因此能够查询某个应用的流量统计信息（不考虑shareuid）。非常方便的是，它的使用不需要特别的权限。另一方面它也一些限制：

（1）无法获取应用的数据流量消耗

从文档中仅能获取到指定uid的流量，但无法区分不同网络类型下的消耗

间接方法是通过监听网络切换，做好流量记录（但是要保证你的应用一直存活，且一定准确接收到网络切换信息），基本不可用。

（2）无法获取某个时间段内的流量消耗

从API文档中看，函数参数没有与时间相关的信息。而且重要的一点是，TrafficStats类中记录的是设备重启以来的流量统计信息。因为TrafficStats 类，底层还是读取/proc/net/xt\_qtaguid/stats 对内容进行解析，将得到对应的结果返回上层。

### NetworkStatsManager

在Android 6.0（API23）中新增加的类，提供网络使用历史统计信息，同时特别强调了可查询指定时间间隔内的统计信息。看看部分函数（非静态）：

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. //查询指定网络类型在某时间间隔内的总的流量统计信息NetworkStats.Bucket querySummaryForDevice(int networkType, String subscriberId, long startTime, long endTime) //查询某uid在指定网络类型和时间间隔内的流量统计信息NetworkStats queryDetailsForUid(int networkType, String subscriberId, long startTime, long endTime, int uid)
3. //查询指定网络类型在某时间间隔内的详细的流量统计信息（包括每个uid）NetworkStats queryDetails(int networkType, String subscriberId, long startTime, long endTime)

从上述函数和文档看，NetworkStatsManager类克服了TrafficStats的查询限制，而且统计信息也不再是设备重启以来的数据。但它也有自己的限制和缺点。   
（1）权限限制   
NetworkStatsManager的使用需要额外的权限，”android.permission.PACKAGE\_USAGE\_STATS”是系统权限，需要主动引导用户开启应用的“有权查看使用情况的应用”（使用记录访问权限）权限，后面会有代码示例。   
（2）文档不完善   
不好说是文档不全，还是我没找对。首先文档中没有给出类的实例对象的构造方法，一开始还是反射获取的，后来才发现可以通过获取系统服务方式得到。另外queryDetailsForUid函数中设置的时间间隔不太有用，没能及时的获取流量统计信息，而是有两个小时的时间间隔。还好可以在querySummary函数中获得。

代码示例

下面说说具体的使用和代码，使用前必须明确的是这里的统计信息都是在网络层以上的数据。   
1.权限设置   
（1）AndroidManifest中添加权限声明

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. <uses-permission android:name="android.permission.READ\_PHONE\_STATE" /><uses-permission android:name="android.permission.PACKAGE\_USAGE\_STATS" tools:ignore="ProtectedPermissions"/>

（2）代码中主动引导用户开启权限   
这里没有说明READ\_PHONE\_STATE的主动获取，大家根据自己的targetSdkVersion设置

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. **private** boolean hasPermissionToReadNetworkStats() {**if** (Build.VERSION.SDK\_INT < Build.VERSION\_CODES.M) {**return** true;
2. }final AppOpsManager appOps = (AppOpsManager) getSystemService(Context.APP\_OPS\_SERVICE);int mode = appOps.checkOpNoThrow(AppOpsManager.OPSTR\_GET\_USAGE\_STATS,
3. android.os.Process.myUid(), getPackageName());**if** (mode == AppOpsManager.MODE\_ALLOWED) {**return** true;
4. }
6. requestReadNetworkStats();**return** false;
7. }// 打开“有权查看使用情况的应用”页面private void requestReadNetworkStats() {
8. Intent intent = **new** Intent(Settings.ACTION\_USAGE\_ACCESS\_SETTINGS);
9. startActivity(intent);
10. }

2.查看设备和某应用的流量统计   
（1）获取NetworkStatsManager示例对象

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. NetworkStatsManager networkStatsManager = (NetworkStatsManager) getSystemService(NETWORK\_STATS\_SERVICE);

（2）查询设备总的流量统计信息

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. NetworkStats.Bucket bucket = null;// 获取到目前为止设备的Wi-Fi流量统计
2. bucket = networkStatsManager.querySummaryForDevice(ConnectivityManager.TYPE\_WIFI, "", 0, System.currentTimeMillis());Log.i("Info", "Total: " + (bucket.getRxBytes() + bucket.getTxBytes()));

（3）查询某应用（uid）的数据流量统计信息

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. // 获取subscriberId
2. TelephonyManager tm = (TelephonyManager) getSystemService(TELEPHONY\_SERVICE);String subId = tm.getSubscriberId();NetworkStats summaryStats;long summaryRx = 0;long summaryTx = 0;NetworkStats.Bucket summaryBucket = **new** NetworkStats.Bucket();long summaryTotal = 0;summaryStats = networkStatsManager.querySummary(ConnectivityManager.TYPE\_MOBILE, subId, getTimesMonthmorning(), System.currentTimeMillis());**do** {
3. summaryStats.getNextBucket(summaryBucket);int summaryUid = summaryBucket.getUid();**if** (uid == summaryUid) {
4. summaryRx += summaryBucket.getRxBytes();summaryTx += summaryBucket.getTxBytes();}
5. Log.i(MainActivity.**class**.getSimpleName(), "uid:" + summaryBucket.getUid() + " rx:" + summaryBucket.getRxBytes() +" tx:" + summaryBucket.getTxBytes());summaryTotal += summaryBucket.getRxBytes() + summaryBucket.getTxBytes();} **while** (summaryStats.hasNextBucket());

3.附赠实用函数   
（1）应用包名查uid

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. NetworkStats summaryStats;long summaryRx = 0;long summaryTx = 0;NetworkStats.Bucket summaryBucket = **new** NetworkStats.Bucket();long summaryTotal = 0;summaryStats = networkStatsManager.querySummary(ConnectivityManager.TYPE\_MOBILE, subId, getTimesMonthmorning(), System.currentTimeMillis());**do** {
2. summaryStats.getNextBucket(summaryBucket);int summaryUid = summaryBucket.getUid();**if** (uid == summaryUid) {
3. summaryRx += summaryBucket.getRxBytes();summaryTx += summaryBucket.getTxBytes();}
4. Log.i(MainActivity.**class**.getSimpleName(), "uid:" + summaryBucket.getUid() + " rx:" + summaryBucket.getRxBytes() +" tx:" + summaryBucket.getTxBytes());summaryTotal += summaryBucket.getRxBytes() + summaryBucket.getTxBytes();} **while** (summaryStats.hasNextBucket());

（2）获得本月第一天0点时间

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. **public** **static** long getTimesMonthMorning() {
2. Calendar cal = Calendar.getInstance();cal.set(cal.get(Calendar.YEAR), cal.get(Calendar.MONTH), cal.get(Calendar.DAY\_OF\_MONTH), 0, 0, 0);cal.set(Calendar.DAY\_OF\_MONTH, cal.getActualMinimum(Calendar.DAY\_OF\_MONTH));**return** cal.getTimeInMillis();}

4.提示无权限信息

**[源码]** [view plain](http://www.php361.com/index.php?c=index&a=view&id=1695)

1. 15:39:06.531 5276-5276/cn.arainfo.test.android.testapp1 E/AndroidRuntime: FATAL EXCEPTION: mainProcess: cn.arainfo.test.android.testapp1, PID: 5276java.lang.SecurityException: Network stats history of uid 10145 is forbidden **for** caller 10144at android.os.Parcel.readException(Parcel.java:1665)
2. at android.os.Parcel.readException(Parcel.java:1618)
3. at android.net.INetworkStatsSession$Stub$Proxy.getHistoryIntervalForUid(INetworkStatsSession.java:425)
4. at android.app.usage.NetworkStats.startHistoryEnumeration(NetworkStats.java:433)
5. at android.app.usage.NetworkStatsManager.queryDetailsForUidTag(NetworkStatsManager.java:254)
6. at android.app.usage.NetworkStatsManager.queryDetailsForUid(NetworkStatsManager.java:219)

统计测试

（1）测试设备   
小米5S Plus  Android 6.0  和 华为Mate9 Android 7.0设备上实际测试   
（2）流量差距   
实际测试流量有30M左右，和运营商流量统计相差2M左右

### REF

[Android应用流量统计——NetworkStatsManager使用](http://www.php361.com/index.php?c=index&a=view&id=1695)

# 源码分析

## TrafficStats

Framework/base/core/java/android/net/TrafficStats.java

最后都是调用 系统 Native

核心方法：

TrafficStats 系统封装的流量统计类，适配版本 Android 2.2 以上，主要用于流量统计的方法有：

getMobileRxBytes -- 获得 mobile 的接受流量

getMobileTxBytes -- 获得 mobile 的发送流量

getTotalRxBytes -- 获得总共的接受流量（ mobile + wifi ）

getTotalTxBytes -- 获得总共的发送流量（ mobile + wifi ）

getUidRxBytes -- 获得指定 uid 的接受流量（ mobile + wifi ）

getUidTxBytes -- 获得指定 uid 的发送流量（ mobile + wifi ）

NetworkStatsManager 一个强大的流量统计工具，适配版本 android 6.0，需要系统权限才能使用，不实用

查看源码

public static long getUidRxBytes(int uid) {

// This isn't actually enforcing any security; it just returns the

// unsupported value. The real filtering is done at the kernel level.

final int callingUid = android.os.Process.myUid();

if (callingUid == android.os.Process.SYSTEM\_UID || callingUid == uid) {

return nativeGetUidStat(uid, TYPE\_RX\_BYTES);

} else {

return UNSUPPORTED;

}

}

可以看到在获取流量时，其实是在调用 nativeGetUidStat(uid, TYPE\_RX\_BYTES)，这个方法是系统 Native 方法，其实 TrafficStats 这个类所有的方法都是在调用三个 Native 方法：

private static native long nativeGetTotalStat(int type);

private static native long nativeGetIfaceStat(String iface, int type);

private static native long nativeGetUidStat(int uid, int type);

Native 方法就需要去下载 Android 源码进行查看了，在源码中找到

## android\_net\_TrafficStats.cpp

在 android\_net\_TrafficStats.cpp 查看 gMethods 内 nativeGetUidStat 在源码中对应的方法为 getUidStat

static const JNINativeMethod gMethods[] = {

{"nativeGetTotalStat", "(I)J", (void\*) getTotalStat},

{"nativeGetIfaceStat", "(Ljava/lang/String;I)J", (void\*) getIfaceStat},

{"nativeGetUidStat", "(II)J", (void\*) getUidStat},

};

### getUidStat

查看 getUidStat 的源码，其实通过 parseUidStats(uid, &stats) 解析数据，通过 getStatsType(&stats, (StatsType) type) 读取数据

static jlong getUidStat(JNIEnv\* env, jclass clazz, jint uid, jint type) {

struct Stats stats;

memset(&stats, 0, sizeof(Stats));

if (parseUidStats(uid, &stats) == 0) {

return getStatsType(&stats, (StatsType) type);

} else {

return UNKNOWN;

}

}

### parseUidStats(uid, &stats)

解析数据源码：

static const char\* QTAGUID\_UID\_STATS = "/proc/net/xt\_qtaguid/stats";

static int parseUidStats(const uint32\_t uid, struct Stats\* stats) {

FILE \*fp = fopen(QTAGUID\_UID\_STATS, "r");

if (fp == NULL) {

return -1;

}

char buffer[384];

char iface[32];

uint32\_t idx, cur\_uid, set;

uint64\_t tag, rxBytes, rxPackets, txBytes, txPackets;

while (fgets(buffer, sizeof(buffer), fp) != NULL) {

if (sscanf(buffer,

"%" SCNu32 " %31s 0x%" SCNx64 " %u %u %" SCNu64 " %" SCNu64

" %" SCNu64 " %" SCNu64 "",

&idx, iface, &tag, &cur\_uid, &set, &rxBytes, &rxPackets,

&txBytes, &txPackets) == 9) {

if (uid == cur\_uid && tag == 0L) {

stats->rxBytes += rxBytes;

stats->rxPackets += rxPackets;

stats->txBytes += txBytes;

stats->txPackets += txPackets;

}

}

}

if (fclose(fp) != 0) {

return -1;

}

return 0;

}

可以看到源码其实就是打开 "/proc/net/xt\_qtaguid/stats" 按照你需要的字段进行读取数据，看来最重要的就是这个 "/proc/net/xt\_qtaguid/stats" 文件了，那就在读取一下这个文件看一下内容是什么，直接读取文件内容如下：

idx iface acct\_tag\_hex uid\_tag\_int cnt\_set rx\_bytes rx\_packets tx\_bytes tx\_packets rx\_tcp\_bytes rx\_tcp\_packets rx\_udp\_bytes rx\_udp\_packets rx\_other\_bytes rx\_other\_packets tx\_tcp\_bytes tx\_tcp\_packets tx\_udp\_bytes tx\_udp\_packets tx\_other\_bytes tx\_other\_packets

2 rmnet\_data0 0x0 10224 0 98175 208 33887 266 98175 208 0 0 0 0 33887 266 0 0 0 0

3 rmnet\_data0 0x0 10224 1 78165 148 29214 143 78165 148 0 0 0 0 29214 143 0 0 0 0

4 wlan0 0x0 10224 0 0 0 1560 26 0 0 0 0 0 0 1560 26 0 0 0 0,

5 wlan0 0x0 10224 1 573000 629 56692 499 573000 629 0 0 0 0 56692 499 0 0 0 0

可以看到，第一行数据是表头数据，剩下的就是对应的流量数据，还可以通过 iface 进行区分 net 还是 wlan ，这正是我们需要的数据。

# 核心

systemUI:(getTotalTxPackets)

Settings:

## Ref

[SoftapCmd命令](https://book.2cto.com/201405/43241.html)

[**wifi热点移植总结**](http://huangqinqin.iteye.com/blog/1286084)

[Android 4.1 Netd详细分析（一）概述与应用实例](http://blog.csdn.net/xiaokeweng/article/details/8130218)

<http://gaozhipeng.me/posts/nativedaemonconnector_source_code/>

# Task

<https://www.google.com.hk/search?safe=strict&ei=aFddWtDTNomp0AS0j4Vo&q=android+TrafficStats+%E6%BA%90%E7%A0%81&oq=android+TrafficStats+%E6%BA%90%E7%A0%81&gs_l=psy-ab.3...900205.900205.0.900580.1.1.0.0.0.0.109.109.0j1.1.0....0...1c.1.64.psy-ab..0.0.0....0.bWU8TyS4qfs>

<http://blog.csdn.net/goohong/article/details/7937148>

http://blog.csdn.net/DaHeMaDeGeGe/article/details/51259129