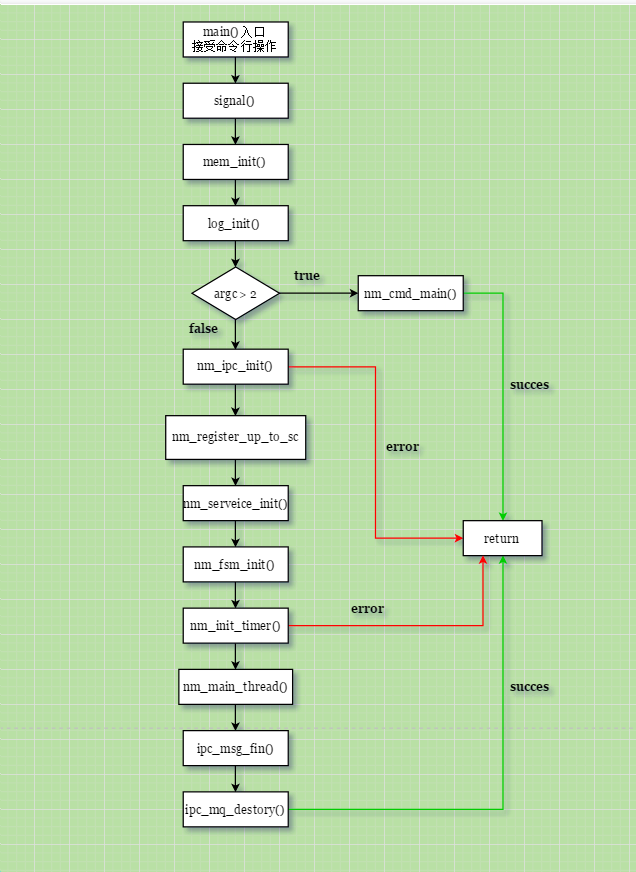
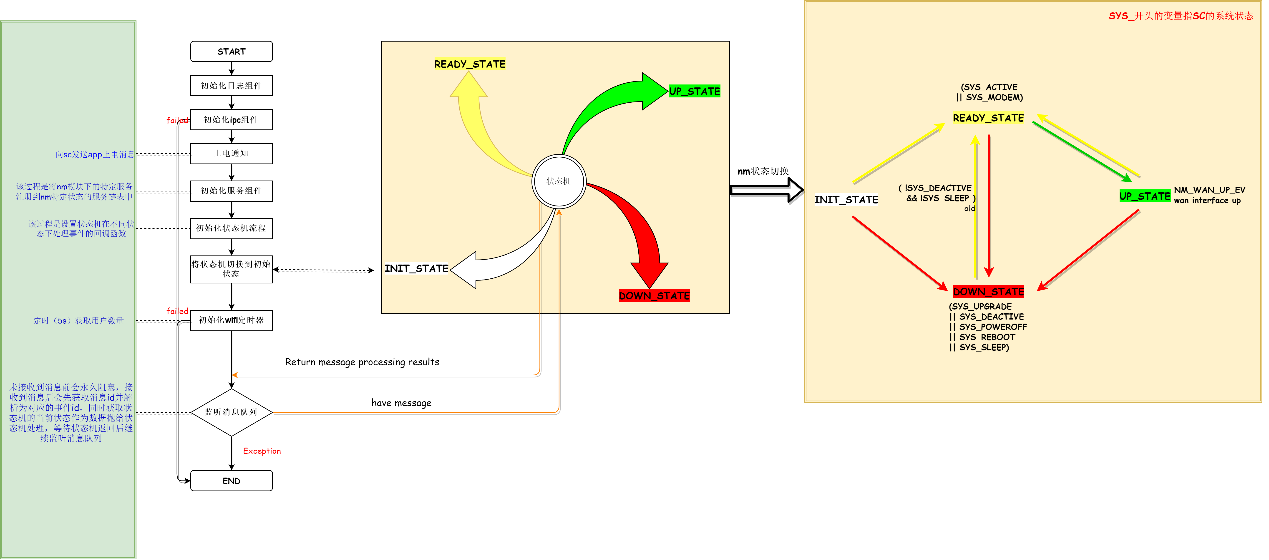
* NM代码流程



* NM运行示意图



* NM模块梳理

*忽略命令行模块。*

NM模块可以按照如下几块：

1. 组件初始化

a. 内存、日志、 IPC

1. 上电通知
2. 发送 异步-app上电通知-NM 消息到SC
3. 服务模块
4. 设计和实现NM模块的服务组件 （start、 stop、 restart）
5. 将各服务注册到指定状态的储存链表中进行控制
6. 状态机（fsm）模块
7. 设计和实现不同状态下处理 ’事件’ 的回调函数
8. 接收处理 ’事件’ 的接口
9. WIFI定时器
10. 查询更新用户数量
11. 消息监听
12. 循环监听IPC，获取其他模块发送到NM的消息
13. 将 ‘消息’ 转换为对应事件，调用fsm ‘事件’ 处理接口

* signal(int, \_\_sighandler\_t )

信号处理

signal(SIGCHLD, SIG\_IGN)

SIGCHLD信号产生条件：子进程终止时会向父进程发送SIGCHLD信号，告知父进程回收自己

SIG\_DFL ：默认的处理方式是不理会这个信号，但是也不会丢弃子进行状态，所以如果不用wait，waitpid

对其子进行进行状态信息回收，会产生僵尸进程。  
SIG\_IGN ：忽略的处理方式，这个方式和默认的忽略是不一样的语意，暂且我们把忽略定义为SIG\_IGN，在这种方式下，子进程状态信息会被丢弃，也就是自动回收了，所以不会产生僵尸进程，但是问题也就来了，

wait，waitpid却无法捕捉到子进程状态信息了，如果你随后调用了wait，那么会阻塞到所有的子进程结束，并返

回错误ECHILD，也就是没有子进程等待。

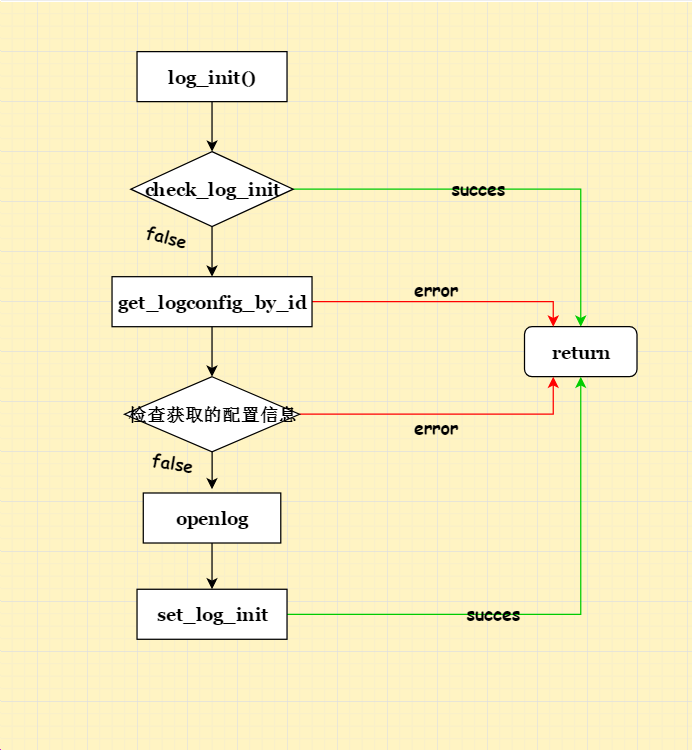
* mem\_init(void (\*oom\_handler)(void))

内存管理模块初始化

无操作

* log\_init(long)

日志模块初始化流程



* check\_log\_init

检查当前模块的日志是否已经初始化 (g\_module\_descrip 长度大于0，已初始化)

* get\_logconfig\_by\_id

1. 首先会通过当前模块的日志ID从全局配置文件（logconfig.c -> g\_code\_config）中获取相关信息 (日志id等价于g\_code\_confiog的索引)
2. 若当前模块的日志id不在全局配置文件中，则在配置文件/usr/bin/logid.cfg中相关信息

* 检查获取的配置信息

检查上一步获取到的配置信息中描述部分的长度是否在有效范围内（1 ~ 32）

typedef struct log\_config\_tag

{

lc\_int32 module\_id;

lc\_char cmd\_arg;

lc\_char module\_descript[32 + 1];

}log\_config\_t

* openlog

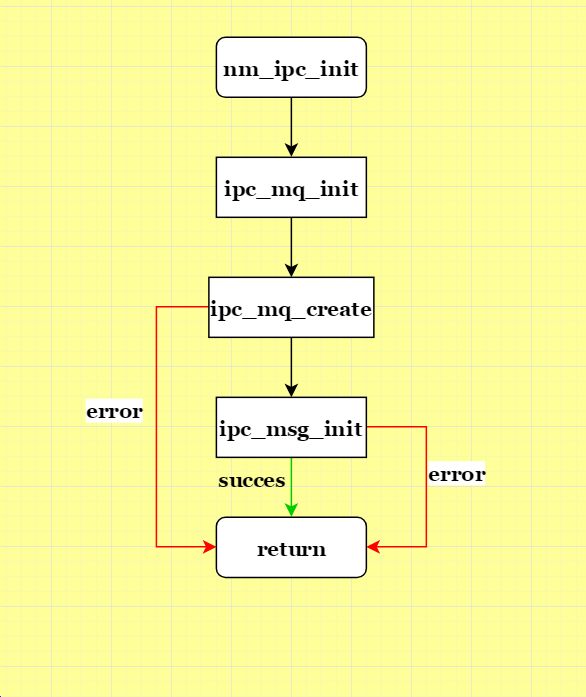
1. 首先将为获取到的配置信息开辟堆内存进行格式化存储”%02d%s” , module\_id, module\_descrip
2. 将配置信息赋值本模块日志属性并与系统日志建立连接

* set\_log\_init

设置log初始化标志g\_module\_descrip

* nm\_cmd\_main(int, char \*\*)
* nm\_ipc\_init()

通信模块初始化流程



1. ipc\_mq\_init()

初始化全局消息队列管理链表的操作互斥量（锁）

1. ipc\_mq\_create()
2. 创建一个消息队列，消息队列的data包含一个消息队列的控制信息（最大长度， 不同优先级消息的数量， 互斥量（锁）， 投放、获取条件变量，消息队列链表头结点等），
3. 消息队列本身作为链表节点添加到全局消息队列管理链表中
4. ipc\_msg\_init()

初始化当前模块的socket（用于发送和接收消息）， 指定消息队列接收进程间消息

* nm\_register\_up\_to\_sc(const long, size\_t)

注册到sc模块

1. ipc\_get\_addr\_by\_id()

获取sc模块接收普通优先级消息的socket地址

1. 开辟消息发送空间buff，填充消息头（消息类型：app上电通知）， 将模块id作为消息数据异步发送给sc模块

* nm\_service\_init()

网络服务初始化

1. 首先nm模块的sevice分为不同系统状态的服务管理维护链表,

init\_services / down\_services / ready\_services / up\_services

1. nm\_service\_init的功能是将网络中的子服务注册到各个系统状态的服务管理维护链表中
2. service\_register 的功能是将某一模块的服务操作（start、stop、restart）注册到服务管理维护链表中

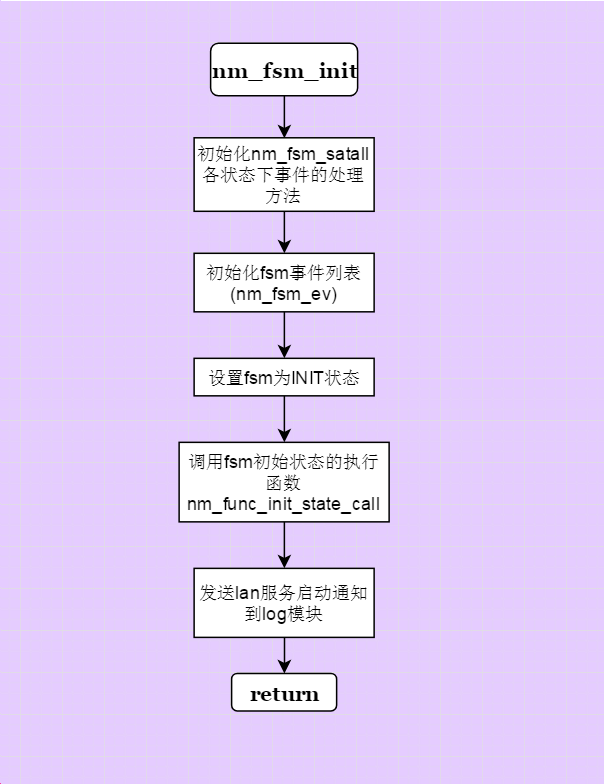
服务注册关联表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 服务\状态 | INIT | DOWN | READY | UP |
| nm\_init | ***reg*** |  |  |  |
| lan | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| dnsmasq | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| named | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| firewall | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| ipa | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| wifidog | ***reg*** | ***reg*** | ***reg*** | ***reg*** |
| rdnssd |  | ***reg*** | ***reg*** | ***reg*** |
| radvd |  |  | ***reg*** | ***reg*** |
| ndp\_proxy |  |  | ***reg*** | ***reg*** |
| redirect |  |  | ***reg*** | ***reg*** |
| igd |  |  | ***reg*** | ***reg*** |
| ntpc |  |  | ***reg*** | ***reg*** |
| ddns |  |  | ***reg*** | ***reg*** |
| easycwmp |  |  | ***reg*** | ***reg*** |
| bw |  |  |  | ***reg*** |
| rmfc |  |  |  | ***reg*** |

|  |  |
| --- | --- |
| 服务 | 说明 |
| nm\_init | 创建并初始化网桥br0，添加相关接口dummy0、rndis0、RJ45等，启动（ifconfig <interface> up）br0及相关接口。 |
| lan | 设置主机名、设置br0的ip地址和子网掩码、设置主机的/tmp/hosts内容 |
| dnsmasq | 创建dnsmasq的配置文件（包含自定义url访问、dns服务、dhcp服务）、利用dnsmasq使其生效 |
| Named (\*) | Nmbd |
| firewall | 通过iptables /ip6tables设置ipv4、ipv6防火墙 |
| ipa | 更新配置文件（/etc/IPACM\_cfg.xml）中的lan ip和netmask，更新（/etc/mobileap\_firewall.xml）并同步配置文件中相关信息  （目前还不清楚这些文件的作用） |
| Wifidog (\*) | Wifidog |
| rdnssd | 通过rdnssd获取ipv6 dns信息 |
| radvd | Radvd 对于无状态自动配置的ipv6测试需要搭建radvd服务器，这样在路由器/设备发送RS请求(icmpv6 type133)的时候，radvd服务器就可以返回RA消息(icmpv6 type134)，告诉设备全局地址的前缀，设备自己再结合接口ID算出一个可聚集全局单播地址。  dhcp6s是 ipv6的dhcp服务 |
| ndp\_proxy (\*) | NDP（Neighbor Discovery Protocol，邻居发现协议）是IPv6的一个关键协议，它组合了IPv4中的ARP、ICMP路由器发现和ICMP重定向等协议，并对它们作了改进。作为IPv6的基础性协议，NDP还提供了前缀发现、邻居不可达检测、重复地址监测、地址自动配置等功能 |
| redirect | 清除/tmp下指定目标的redirect目录（gps || Bluetooth || traffic || nm）-> 重启firewall、dnsmasq服务 -> nslookup查询域名解析 |
| igd | 在upnp功能开启且网络接口启动的情况下，设置upnp的配置信息，并调用miniupnpd使其生效 |
| ntpc (\*) | 利用ntpclient工具定时到指定ntp服务器获取信息来更新本地设备时间 |
| ddns | ddns客户端工具inadyn，  DDNS（Dynamic Domain Name Server）是动态域名服务的缩写。DDNS是将用户的动态IP地址映射到一个固定的域名解析服务上，用户每次连接网络的时候客户端程序就会通过信息传递把该主机的动态IP地址传送给位于服务商主机上的服务器程序，服务器程序负责提供DNS服务并实现动态域名解析。 |
| easycwmp | 通过ubusd  Easycwmpd  easycwmp是基于cwmp（CPE WAN Management Protocol）协议开发出的CPE（Customer premises equipment）客户端 |
| bw | 利用tc进行流量控制 |
| rmfc |  |

* nm\_fsm\_init()

状态机初始化流程



nm\_set\_fsmst(NM\_NET\_INIT\_STAT)

在进程控制块的属性中设置nm当前状态为INIT状态

nm\_func\_init\_state\_call()

1. 更新当前nm管理维护链表为init\_services
2. 关闭所有wan网络interface
   1. 执行/etc/nm\_script.sh脚本关闭ipv4、ipv6
   2. 从临时nv中获取nm\_v6\_wan\_up\_ifname，删除/tmp/下对应文件的连接，刷新ipv6地址，将ipv6的相关nv清空， 停止radvd和ndp\_proxy服务，
   3. 从临时nv获取nm\_wan\_up\_ifname, 若支持MTU（最大传输单元）， 则设置MTU （ifcongfig <wan interface name> mtu <字节>），删除/tmp/下对应文件的连接，删除路由，将ipv4的相关nv清空
   4. 重启rmfc服务
   5. 重启firewall服务
   6. 重启wifidog服务
   7. 重启igd服务
   8. 重启ntpc服务
   9. 重启bw服务
3. 启动nm、lan、dnsmasq 服务
4. 获取DM模块通信地址，向DM发送usb设备变化通知、底座设备变化通知、温度变化通知。
5. 获取MS模块通信地址，向MS发送主机同步结果通知

（注：发送的是同步注册类型消息）

nm\_lan\_up\_notify()

获取LOG模块地址，向LOG模块发送lan服务启动通知

定义

// nm状态定义

enum nm\_stat {

NM\_NET\_INIT\_STAT = 0, // init

NM\_NET\_DOWN\_STAT, // down

NM\_NET\_READY\_STAT, // ready

NM\_NET\_UP\_STAT, // up

NM\_NET\_MAX\_STAT, // max value

};

// nm 事件定义

enum nm\_event{

NM\_MOD\_CHANGE\_EV=0,

NM\_WIFI\_TIMEOUT,

NM\_USB\_MODE\_EV,

NM\_WAN\_UP\_EV,

NM\_WAN\_DOWN\_EV,

NM\_MANAGER\_DO\_EV,

NM\_STOP\_NOTIFY\_EV，

NM\_CHARGER\_STAT\_EV,

NM\_GET\_USER\_NUM\_EV,

NM\_WDTIMER\_OUT\_EV,

NM\_WIFI\_BUTTON\_EV,

NM\_WIFI\_GET\_WPS\_EV,

NM\_WM\_IP\_UPDOWN\_EV,

NM\_SET\_WORK\_MODE\_EV,

NM\_MOD\_PRE\_CHANGE\_EV,

NM\_TEMP\_CHANGE\_EV,

NM\_LAN\_UP\_EV,

NM\_LAN\_DOWN\_EV,

MS\_SYNC\_RESULT\_EV,

NM\_CRADLE\_PLUG\_EV,

NM\_CONFIG\_UPDATE\_EV,

NM\_EVENT\_MAX\_EV,

}

// 状态机结构

typedef struct nm\_fsm\_stat{

lc\_int32 (\*func)(lc\_int8\*);

lc\_int32 next\_stat;

}nm\_fsm\_stat\_t;

// 状态机数组 [一级索引nm状态][二级索引nm事件]

static nm\_fsm\_stat\_t nm\_fsm\_statall[NM\_NET\_MAX\_STAT][NM\_EVENT\_MAX\_EV];

状态事件-处理关系表 （nm\_fsm\_statall）

|  | init | dwon | ready | up | **(func)** |
| --- | --- | --- | --- | --- | --- |
| NM\_MOD\_CHANGE\_EV | **🡪** | 🡪 | **🡪** | **🡪** | ***nm\_mode\_change*** |
| NM\_WIFI\_TIMEOUT |  |  |  |  |  |
| NM\_USB\_MODE\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_rev\_usb\_notify*** |
| NM\_WAN\_UP\_EV |  |  | **🡪** | **🡪** | ***nm\_api\_ready\_wanup*** |
| NM\_WAN\_DOWN\_EV |  |  |  | **🡪** | ***nm\_up\_wandown*** |
| NM\_MANAGER\_DO\_EV | **🡪** | 🡪 | **🡪** | **🡪** | ***nm\_manager\_service\_do*** |
| NM\_STOP\_NOTIFY\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_stop\_main\_thread*** |
| NM\_CHARGER\_STAT\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_rev\_lc\_charger\_notify*** |
| NM\_GET\_USER\_NUM\_EV | **🡪** | 🡪 | **🡪** | **🡪** | ***nm\_get\_user\_num*** |
| NM\_WDTIMER\_OUT\_EV |  |  |  |  |  |
| NM\_WIFI\_BUTTON\_EV |  |  |  |  |  |
| NM\_WIFI\_GET\_WPS\_EV |  |  |  |  |  |
| NM\_WM\_IP\_UPDOWN\_EV |  |  |  |  |  |
| NM\_SET\_WORK\_MODE\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_set\_work\_mode*** |
| NM\_MOD\_PRE\_CHANGE\_EV |  | 🡪 |  |  | ***nm\_mode\_change*** |
| NM\_TEMP\_CHANGE\_EV |  |  |  | **🡪** | ***nm\_temp\_change*** |
| NM\_LAN\_UP\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_api\_ready\_lanup*** |
| NM\_LAN\_DOWN\_EV |  | 🡪 | **🡪** | **🡪** | ***nm\_api\_ready\_landown*** |
| MS\_SYNC\_RESULT\_EV |  |  | **🡪** | **🡪** | ***nm\_ms\_sync\_result\_handle*** |
| NM\_CRADLE\_PLUG\_EV |  |  | **🡪** | **🡪** | ***nm\_cradle\_plug*** |
| NM\_CONFIG\_UPDATE\_EV |  |  | **🡪** | **🡪** | ***nm\_network\_config\_via\_cradle*** |
| NM\_EVENT\_MAX\_EV |  |  |  |  |  |

事件处理详解表

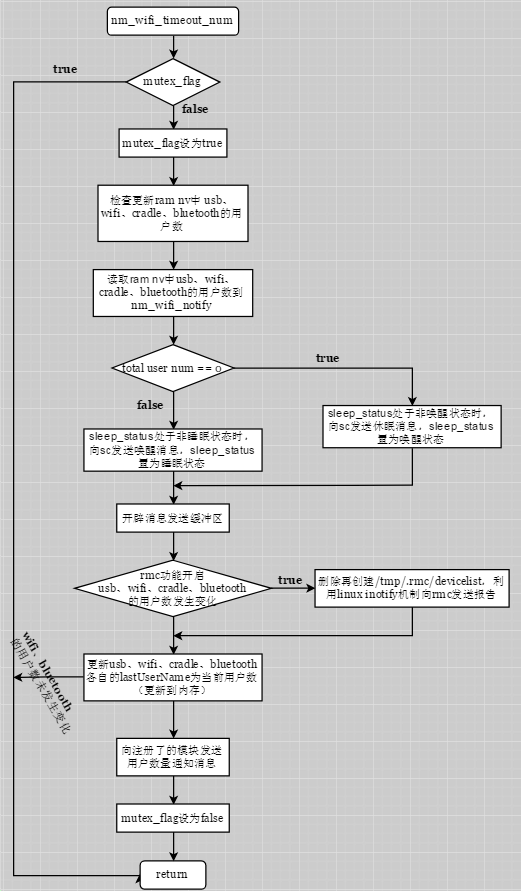
| **(func)** | **状态机回调函数** |
| --- | --- |
| ***nm\_mode\_change*** |  |
|  |  |
| ***nm\_rev\_usb\_notify*** |  |
| ***nm\_api\_ready\_wanup*** |  |
| ***nm\_up\_wandown*** |  |
| ***nm\_manager\_service\_do*** |  |
| ***nm\_stop\_main\_thread*** |  |
| ***nm\_rev\_lc\_charger\_notify*** |  |
| ***nm\_get\_user\_num*** |  |
|  |  |
|  |  |
|  |  |
|  |  |
| ***nm\_set\_work\_mode*** |  |
| ***nm\_mode\_change*** |  |
| ***nm\_temp\_change*** |  |
| ***nm\_api\_ready\_lanup*** |  |
| ***nm\_api\_ready\_landown*** |  |
| ***nm\_ms\_sync\_result\_handle*** |  |
| ***nm\_cradle\_plug*** |  |
| ***nm\_network\_config\_via\_cradle*** |  |
|  |  |

* nm\_init\_timer()

初始化WIFI定时器

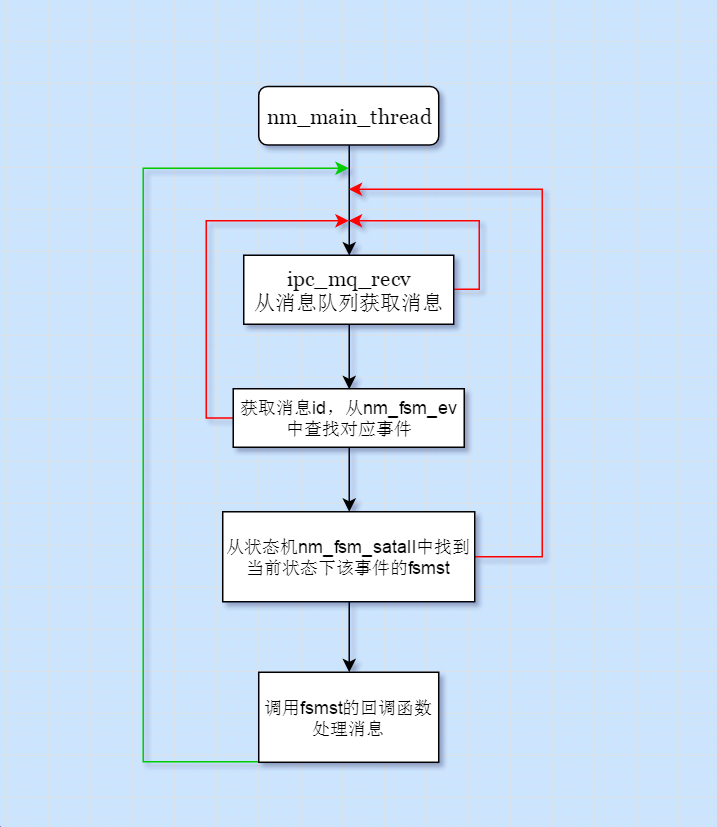
1. 初始化定时器的后台线程及全局管理链表等
2. 创建目录 /tmp/.devlist /tmp/.maclist
3. 创建名为wifi\_timer 时间间隔为5s的周期性wifi定时任务
4. 启动wifi\_timer定时任务

wifi定时任务的回调函数

nm\_wifi\_timeout\_num 

* nm\_main\_thread()

主线程任务



消息-事件关系表

| 事件ID | 消息id |
| --- | --- |
| NM\_MOD\_CHANGE\_EV | SC\_STATE\_CHANGE\_REQ /\* 状态切换请求 \*/ |
| NM\_WIFI\_TIMEOUT, |  |
| NM\_USB\_MODE\_EV, | DM\_USB\_PLUG\_NOTIFY /\* USB设备变化通知 \*/ |
| NM\_WAN\_UP\_EV, | NM\_MSG\_WAN\_UP\_REQ |
| NM\_WAN\_DOWN\_EV | NM\_MSG\_WAN\_DOWN\_REQ |
| NM\_MANAGER\_DO\_EV | NM\_PWR\_CHARGER\_TYPE\_NOTIFY /\* 电源变化 \*/ |
| NM\_STOP\_NOTIFY\_EV | SC\_POWEROFF\_REQ /\* 关电请求 \*/ |
| NM\_CHARGER\_STAT\_EV |  |
| NM\_GET\_USER\_NUM\_EV |  |
| NM\_WDTIMER\_OUT\_EV |  |
| NM\_WIFI\_BUTTON\_EV |  |
| NM\_WIFI\_GET\_WPS\_EV |  |
| NM\_WM\_IP\_UPDOWN\_EV |  |
| NM\_SET\_WORK\_MODE\_EV | SC\_SET\_WORK\_MODE\_REQ /\* 设置系统工作模式请求 \*/ |
| NM\_MOD\_PRE\_CHANGE\_EV | SC\_PRE\_STATE\_CHANGE\_REQ /\* 预览状态变化请求 \*/ |
| NM\_TEMP\_CHANGE\_EV | DM\_TEMP\_CHANGE\_NOTIFY /\* 温度变化通知 \*/ |
| NM\_LAN\_UP\_EV | NM\_MSG\_LAN\_UP\_NOTIFY |
| NM\_LAN\_DOWN\_EV | NM\_MSG\_LAN\_DOWN\_NOTIFY |
| MS\_SYNC\_RESULT\_EV | MS\_SYNC\_RESULT\_NOTIFY /\* 主机同步结果通知 \*/ |
| NM\_CRADLE\_PLUG\_EV | DM\_CRADLE\_PLUG\_NOTIFY /\* 底座设备变化通知\*/ |
| NM\_CONFIG\_UPDATE\_EV | NM\_MSG\_CONFIG\_UPDATE\_REQ /\* nm配置更新请求 \*/ |
| NM\_EVENT\_MAX\_EV | nm事件定义的上边界 |

* ipc\_msg\_fin()

清理进程间通信资源

1. 检查初始化标志（未初始化直接返回）
2. 向本模块高优先级的socket异步发送<消息接收线程停止>的通知
3. 等待消息接收线程结束
4. 关闭本模块所有socket并将置空管理数组
5. 销毁注册通知链表
6. 销毁同步等待链表
7. 销毁各链表管理节点的互斥量（锁）
8. 设置初始化标志为FALSE

* ipc\_mq\_destory()

销毁线程间通信资源

从消息队列管理节点获取消息队列的头结点， 销毁消息队列