**雷达IP管理方案与实现**

唐松泉

2023/9/6

修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 责任人 | 时间 | 说明 |
| V1.0 | 唐松泉 | 2023/9/6 | 初版 |
|  |  |  |  |
|  |  |  |  |

目录

[1 文档背景 4](#_Toc144970813)

[2 文档面向对象 4](#_Toc144970814)

[3 需求实现设计 4](#_Toc144970815)

[4 实现方案概览 5](#_Toc144970816)

[4.1 网络模块初始化 5](#_Toc144970817)

[4.2 本地自动检测配置 6](#_Toc144970818)

[4.2.1 自动检测线程 6](#_Toc144970819)

[4.2.2 网络配置脚本 7](#_Toc144970820)

[5 技术实现要点说明 8](#_Toc144970821)

[5.1 网络配置 8](#_Toc144970822)

[5.2 判断物理网线断开 9](#_Toc144970823)

[5.3 判断配置成功与否 10](#_Toc144970824)

[6 附录 11](#_Toc144970825)

# 1 文档背景

雷达在现有功能基础上，提出了IP自动管理的功能需求，即在拔插电脑或者路由时，都能自动的根据用户的配置，配置雷达的IP地址，保持雷达一直有IP在线。

本文档基于雷达Linux版本现有的IP管理需求，提出的一种实现方案和技术相关实现细节，旨在帮助使用和阅读的相关人员理解管理的方式和原理。

# 2 文档面向对象

所有研发人员。

# 3 需求实现设计

根据现有需求，提出以下配置流程：在出现雷达网络变动（网线拔插等情况），如果当前用户配置的是使用dhcp，则优先用dhcp自动获取IP，获取失败情况下，返回使用用户配置好的静态IP。

例如，当用户插入路由，此时路由如果有dhcp，则优先用dhcp获取IP，如果此时用户换插回电脑，在dchp获取IP失败情况下，则使用静态IP，默认是192.168.235.55.



fig1 实现设计流程

# 4 实现方案概览

下图是整体方案模块概览图，方案实现了一个统一配置网卡的脚本，任何配置途径都使用该脚本进行配置；配置网络途径主要有“网络配置模块初始化”、“用户远程配置”和“本地检测自动配置”，其中“网络配置模块初始化”指在程序加载之后，根据当前用户使用的配置和物理网络情况初始化的网络；“用户远程配置”指用户可以通过远程来配置当前雷达的网络，“本地检测自动配置”程序实时检查网络的变化（例如网线拔插），根据当前用户配置，进行网络配置。



fig2 方案概览

## 4.1 网络模块初始化

网络管理功能初始化优先于其他网络通信操作，初始化流程如下：



fig3 网络管理功能初始化

初始化过程中，配置文件是保存用户当前的配置，包括ip和获取ip方式（静态或者dhcp），网络配置脚本是一次性写，即第一次运行程序时脚本不存在则写脚本文件，后面重新运行程序，除非文件被人为删除或者文件系统异常，否则文件一直存在则不会进行文件更新。初始化过程还对网络进行初始化，根据当前用户选择的配置进行网络配置，例如当前用户保存在配置文件的配置是dhcp，静态ip是192.168.235.22，则初始化网络首先会用dhcp客户自动获取ip，当ip获取失败时，再进行静态ip的配置。

初始化最后的流程是启动本地自动检测的线程，该线程是用来检查网络断开于链接情况，根据情况，读取用户配置再进行网络配置，详细参考下一节。

## 4.2 本地自动检测配置

本地自动检测将实时调用网络配置脚本进行配置。

### 4.2.1 自动检测线程

在初始化最后，会启动本地自动检测配置的线程，该线程流程图如下：



fig4 自动检测线程

线程实时获取网卡激活与否的事件，当网线拔插后者认为的启动/停止网络服务时，内核会上报网络up/down事件，在配置完网络后，需要重启网络生效，此时在线程获取事件里需要过滤程序重启网络的事件。

在收到网卡下线(down事件)后，检查网线连接情况，网线断开时记录下当前状态，在收到网卡上线(up事件)后，检查网线连接和上一次的状态是断开后，可判断当前事件是插入网线，再根据用户的配置进行网络配置。

网络配置时，当用户配置的是dhcp自动获取ip，且此时dhcp获取ip失败时，将返回使用静态ip，该静态ip是用户配置的在静态配置情况下的ip，默认是192.168.235.55.

网络配置时，调用网络配置脚本进行配置；

### 4.2.2 网络配置脚本

网络配置脚本主要目的是修改网卡配置文件(/etc/network/interfaces)，根据用户传进来的配置，来修改该配置文件，完成后在重启网络。

配置脚本流程如下：



fig5 网络配置脚本流程

dhcp获取IP的次数，使用默认的3次超时返回，脚本退出的返回码，是启动网络服务的结果，成功时为0，其他为失败。

# 5 技术实现要点说明

## 5.1 网络配置

网络配置使用的是原始的/etc/network/interfaces，目前推荐的方式netplan，但当前雷达没有移植netplan和service服务，所有选择用/etc/network/interfaces的配置文件来配置网卡，使用/etc/init.d/networking stop/start来实现网卡复位。

程序生成了一个修改/etc/network/interfaces的脚本：

"#!/bin/bash\n \

set -e\n \

RADAR\_NETWORK\_CONFIG\_FILE=/etc/network/interfaces\n \

NETWORK\_MODE=$1\n \

IP=$2\n \

sed -i '/auto eth0/,/iface eth1/{/auto eth0/!{/iface eth1/!d}}' ${RADAR\_NETWORK\_CONFIG\_FILE}\n \

if [ \"${NETWORK\_MODE}\" == \"dhcp\" ]; then\n \

    sed -i '/auto eth0/a iface eth0 inet dhcp' ${RADAR\_NETWORK\_CONFIG\_FILE}\n \

else\n \

    sed -i '/auto eth0/a iface eth0 inet static \\naddress '${IP}'\\nnetmask 255.255.255.0' ${RADAR\_NETWORK\_CONFIG\_FILE}\n \

fi\n \

/etc/init.d/networking stop\n \

sleep 1s\n \

/etc/init.d/networking start\n \

echo -e \"result $?\"\n \

if [ $? -eq 0 ];then\n \

    exit 0\n \

else\n \

    exit 1\n \

fi\n";

## 5.2 判断物理网线断开

Linux下/sys/class/net/eth0/carrier实时记录了当前网线连接状态，该目录下还有其他状态记录的节点，这里使用了carrier，结合Linux的link up和link down事件，来判断当前物理网络情况。

在复位网络时，/sys/class/net/eth0/carrier实际上记录的是复位前的状态，在网卡生效才会更新到最新值，所以在复位之后，需要等待复位完成再去获取状态。同样link up/down事件，在复位时也会发出一次up/down事件，为了过滤该事件，使用在获取到该up/down事件后，再判断网络连接情况，来确认当前网络变化。

static int get\_netlink\_status(void)

{

    char eth\_status[2] = {0};

    int netstatus\_fd = 0;

    int st = -1;

    int retval = 0;

    netstatus\_fd = open(NETLINK\_STATUS\_NODE, O\_RDONLY);

    retval = read(netstatus\_fd, eth\_status, 2);

    if(retval < 0) {

        return retval;

    }

    close(netstatus\_fd);

    st = eth\_status[0]-0x30;

    return st;

}

## 5.3 判断配置成功与否

在执行脚本配置网卡之后，需要判断当前配置是否成功，这里通过获取IP来判断是否成功，当获取IP成功时，认为当前配置成功，否则认为当前网卡的配置失败。

static int check\_config\_result(char \*netcard\_name)

{

    int ret = 0;

    int fd = 0;

    struct ifreq ifreq = {0};

    fd = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_IP);

    memset(&ifreq, 0x00, sizeof(struct ifreq));

    strcpy(ifreq.ifr\_name, netcard\_name);

    ret = ioctl(fd, SIOCGIFADDR, &ifreq);

    if (ret < 0) {

        perror("SIOCGIFADDR: ");

    } else {

        ret = 0;

    }

    return ret;

}

# 6 附录

<https://blog.csdn.net/u012478275/article/details/117353793>

<https://blog.51cto.com/u_11626714/4880873>

<https://www.cnblogs.com/bo083/archive/2012/11/23/2785172.html#:~:text=Linux%20%E7%BD%91%E7%BB%9C%E9%85%8D%E7%BD%AE%201%E3%80%81%E5%B8%B8%E7%94%A8%E9%85%8D%E7%BD%AE%E7%BD%91%E7%BB%9C%E6%8C%87%E4%BB%A4,%281%29%20%E9%85%8D%E7%BD%AEeth0%E7%9A%84IP%E5%9C%B0%E5%9D%80%EF%BC%8C%20%E5%90%8C%E6%97%B6%E6%BF%80%E6%B4%BB%E8%AF%A5%E8%AE%BE%E5%A4%87>