ip 命令手册

摘要

ip 是 iproute2 软件包里面的一个强大的网络配置工具，它能够替代一些传统的网络管理工具。例如：ifconfig、route 等。这个手册将分章节介绍 ip 命令及其选项。

本文的原文在 http://defiant.coinet.com/iproute2/ip-cref/(2002-10-15 18:40:46)

作者：Alexey N.Kuznetsov

编译：nixe0n　整理：pangty

1.关于这篇文档

ip 是 iproute2 软件包里面的一个强大的网络配置工具，本文将分章节介绍 ip 命令及其选项。为了便于理解，作者在本文中列举了很多示例。但是，正如作者所说，这不是一个教程，而是一个使用手册。

2. ip 命令的语法

ip 命令的用法如下：

```
ip [OPTIONS] OBJECT [COMMAND [ARGUMENTS]]
```

其中，OPTIONS 是一些修改 ip 行为或者改变其输出的选项。所有的选项都是以-字符开头，分为长、短两种形式。目前，ip 支持如下选项：

-V,-Version	打印 ip 的版本并退出。
-s,-stats,-statistics	输出更为详尽的信息。如果这个选项出现两次或者多次，输出的信息将更为详尽。
-f,-family	这个选项后面接协议种类，包括：inet、inet6 或者 link，强调使用的协议种类。如果没有足够的信息告诉 ip 使用的协议种类，ip 就会使用默认值 inet 或者 any。link 比较特殊，它表示不涉及任何网络协议。
-4	是-family inet 的简写。
-6	是-family inet6 的简写。
-0	是-family link 的简写。
-o,-oneline	对每行行行行行行行行记录都使用用用用单行行行行行回行行行行行行行行用用用用 等工工工工具处理理
	行行行行行行行行行行输出， 字符代替。 -r,-resolve 如果你需要使用用用用用用用用 wc、grep 理理理理 ip 的输出 会用用用用用用用到这个选项。

查询域名解析系统，用获得的主机名代替主机 IP 地址。

OBJECT 是你要管理或者获取信息的对象。目前 ip 认识的对象包括：

link	网络设备
address	一个设备的协议（IP 或者 IPV6）地址
neighbour	ARP 或者 NDISC 缓冲区条目
route	路由表条目
rule	路由策略数据库中的规则
maddress	多播地址

mroute	多播路由缓冲区条目
tunnel	IP 上的通道

另外，所有的对象名都可以简写，例如：address 可以简写为 addr，甚至是 a。

COMMAND 设置针对指定对象执行的操作，它和对象的类型有关。一般情况下，ip 支持对象的增加(add)、删除(delete)和展示(show 或者 list)。有些对象不支持所有这些操作，或者有其它的一些命令。对于所有的对象，用户可以使用 help 命令获得帮助。这个命令会列出这个对象支持的命令和参数的语法。如果没有指定对象的操作命令，ip 会使用默认的命令。一般情况下，默认命令是 list，如果对象不能列出，就会执行 help 命令。

ARGUMENTS 是命令的一些参数，它们倚赖于对象和命令。ip 支持两种类型的参数：flag 和 parameter。flag 由一个关键词组成；parameter 由一个关键词加一个数值组成。为了方便，每个命令都有一个可以忽略的默认参数。例如，参数 dev 是 ip link 命令的默认参数，因此 ip link ls eth0 等于 ip link ls dev eth0。我们将在后面的章节详细介绍每个命令的使用，命令的默认参数将使用 default 标出。

几乎所有的关键词都可以简写为前几个字母。在交互工作时，简写的方式非常方便，但是我们不建议在脚本中使用简写形式。另外，在讲述过程中，所有的“官方”简写方式都会在文章中列出。

3.ip 的错误信息

由于以下原因，ip 可能会操作失败：

> 命令行语法错误：一个未知的关键词(an unknown keyword)；错误的 IP 地址格式(incorrectly formated IP address)。在这种情况下，ip 会打印出错误信息然后退出，在错误信息中会包含失败的原因。有时 ip 也会打印帮助信息。
>
> 参数不能通过一致性校验。
> 由于用户没有提供足够的信息，造成 ip 无法从参数中编译出内核请求。
> 内核返回某些系统调用的错误。ip 使用 perror(3)输出错误信息，因此输出的错误信息包含一段注释以及系统调用号。
> 内核返回 RTNETLINK 请求错误。这类错误信息以"RTNETLIK answers"开头。

ip 的所有操作都是原子操作。例如，如果 ip 执行失败，它不会系统的任何东西，ip link command 例外，它会修改某些设备参数。

我们无法列出所有的错误信息，尤其是语法错误。不过，错误信息的意思都非常明确。下面，我们列举一些经常出现的错误信息：

内核不支持 netlink(netlink 用于在内核模块和用户之间传递信息)，会出现以下错误信息：

```
Cannot open netlink socket: Invalid value
```

内核不支持 RTNETLINK，会出现以下错误信息：

```
Cannot talk to rtnetlink: Connect refused
Cannot send dump request: Connect refused
```

如果在编译内核时没有配置 CONFIG_IP_MULTIPLE_TABLES 选项。在使用 ip 规则时会出现和下面的信息类似的错误信息：

```
kuznet@kaise $ ip rule list
RTNETLINK error: Invalid argument
dump terminated
```

4.ip link--配置网络设备

对象	link 由网络设备，对应的命令显示以及设备的状态变化组成。
命令	set 和 show(或者 list)

4.1.ip link set--改变设备的属性

缩写：set、s

参数：

dev NAME(default)	指定进行操作的网络设备
up/down	起动 / 关闭设备。 例如：ip link set dev eth0 up
arp on/off	改变网络设备的 NOARP 选项。 如果设备处于 UP 状态，不允许进行这个操作。不过，内核和 ip 都不会对在这种情况下的这个操作进行检查。在设备处于运行状态下改变这个选项会造成无法预料的后果。
multicast on/off	改变网络设备的 MULTICAST 选项。
dynamic on/off	改变网络设备的 DYNAMIC 选项。
name NAME	把设备的名字改为 NAME(例如：eth0)。如果设备处于运行状态或者已经配置了地址，建议不要进行这个操作。
txqueuelen NUMBER 或者 txqlen NUMBER	改变设备传输队列的长度。 例如：ip link set dev eth0 txqueuelen 100
mtu NUMBER	改变网络设备 MTU(最大传输单元)的值。

	例如：ip link set dev eth0 mtu 1500
address LLADDRESS	修改网络设备的MAC地址。 例如：ip link set dev eth0 address 00:01:4f:00:15:f1
broadcat LLADDRESS 或者brd LLADDRESS	修改数据链路层广播地址。 注意：对于大多数的网络设备(例如：以太网)，修改链路层广播地址会对网络造成破坏。因此，如果对此没有很深的理解，最好不要使用这个操作。
peer LLADDRESS	当使用点对点连接时，使用这个操作可以修改对端的数据链路层地址。

注意：ip不能修改PROMISC或者ALLMULTI选项。这两个选项已经比较陈旧，而且也不应该随便修改。

注意：如果同时修改多个参数，任何一个修改失败，ip都会立即取消操作。这种情况可能使系统进入无法预料的状态。为了避免出现这种情况，尽量不要使用ip link set同时修改多个参数，例如：ip link set dev eth0 mtu 1500 txqueuelen 100。

4.2.ip link show--显示设备属性

缩写：show、list、lst、sh、ls、l

参数

dev NAME(default)	NAME指定网络设备名称，例如：eth0。如果省略了这个参数，所有的设备属性就都会被列出。
up	只显示处于活动状态网络接口的信息。

输出格式

```
kuznet@alisa:~ $ ip link ls eth0

3: eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff:

kuznet@alisa:~ $ ip link ls sit0
5: sit0@NOME:  mtu 1480 qdisc noqueue
    link/sit 0.0.0.0 brd 0.0.0.0

kuznet@alisa:~ $ ip link ls dummy
2: dummy:  mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
kuznet@alisa:~ $
```

在显示的信息中，每个引号之前的数字是一个接口索引，用于识别网络接口。这个数字后面是网络接口的名字(例如：eth0、dummy 等)，它也和网络接口一一对应。不过，在某些情况下，例如：驱动模块被卸载，对应的接口名就会从列表中消失，而其它新创建的接口就会使用相同的名字。系统管理员可以 ip link set name 修改接口的名字。

接口名可以是别的,或者是@NONE。这意味着这个设备被绑定到其它的设备，例如数据包被发送到这个设备，由这个设备封装，并从 master 设备发出。如果设备名字是 NONE，就表示 master 设备是未知的。

接着，我们看到的是 mtu(Maximal Transfer Unit,最大传输单元)。它决定这个接口单个数据包能够传输多少数据。

qdisc(queuing discipline)显示这个网络接口使用的排队算法。noqueue 表示不对数据包进行排队；noop 表示这个网络接口出于黑洞模式，也就是所有进入本网络设备的数据会直接被丢弃。qlen 是网络接口传输队列的默认长度。

网络接口可以有如下标志：

UP	这个设备处于运行状态，可以接收、发送数据包。
LOOPBACK	这个接口不能用于和其它的主机通讯，所有发送到这个接口的数据包都会返回，而且这种接口只能接收反弹回来的数据包。
BROADCAST	这个设备具有把数据包发送到所有主机的能力。以太网连接是一个很典型的例子。
POINTTOPOINT	两个节点之间是直接连接的。某个接点发出的所有数据包都会发到对端节点，接收到的所有数据包也都是从对端节点发过来的。
MULTICAST	这个标志表示设备具有多播能力，能够把数据包发送到某些相邻的网络节点。实际上，广播是多播的一个特例，它的多播组包括连接上的所有节点。 从定义上，POINTTOPOINT 和 BROADCAST 连接都属于多播。

*如果网络接口的标志不属于 LOOPBACK、BROADCAST 和 POINTTOPOINT 的任何一个，就假定是 NMBA(Non-Broadcast Multi-Access)类型。这是最为普遍的一个标志。

PROMISC	设备处于混杂模式，接收连接上的所有数据，不管目的地址是否是自己。通常，这种模式主要用于网桥和网络监视。
ALLMULTI	设备接收连接上的所有多播数据包，多播路由器(muliticast router)使用这种模式。

NOARP	这个标志和其它的标志不同，它的含义和涉及的网络协议有关。它一般表示这个设备无需地址解析，软件或者硬件不必借助于系统协议栈的帮助就知道如何把数据包投递到目的地。
DYNAMIC	这个标志表示这个网络接口是动态建立和撤消的。
SLAVE	表示这个接口被绑定到其它的网络接口。

*除此之外，还有其它一些标志。这些标志或者已经过时(例如：NOTRAILERS)，或者还没有实现(如：DEBUG)，或者只是特定于某些设备(例如：MASTER、AUTOMEDIA、PORTSEL)。因此，在此我们不作讨论。

*对于 PROMISC 和 ALLMULTI 标志，ifconfig 和 ip 显示的值是不同的。ip link ls 命令显示的是设备的真正状态，而 ifconfig 显示的是自己设置的虚拟设备状态。

显示信息的第二行包含和链路层地址(MAC 地址)相关的信息。其中，第一个词(ether、sit)定义接口的硬件类型。而接口的硬件类型又决定 MAC 地址的格式和语法。默认的格式是硬件的 MAC 地址和广播地址(如果是点对点连接方式，就是对端的地址)，地址是用冒号隔开的 16 进制数字。不过，默写类型的连接有其特定的地址格式，例如：IP 通道的地址格式是用点分开的 IP 地址。

NBMA(Non-Broadcast Multi-Access)连接没有明确定义的广播地址和对端地址。不过，这个域包含一些有用的信息，例如：倚赖于 ARP 服务器的广播地址。

使用这个命令不会显示多播地址，需要使用 ip maddr ls 命令。详情请参考第 9 节 ip maddr ls。

统计信息

使用-statistics 选项，ip 命令会打印出网络接口的统计信息，例如：

```
kuznet@alisa:~ $ ip -s link ls eth0

3: eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    2449949362 2786187 0       0       0       0
    TX: bytes  packets errors  dropped carrier collsns
    178558497  1783946 332     0       332     35172
kuznet@alisa:~ $
```

RX:和 TX:分别是接收和发送统计信息的开头。得到的统计信息包括：

bytes	网络接口发送或者收到的字节数。如果字节数超过数据类型能够表示的最大数值，就会造成回卷。因此，你如果想连续监视这个指标，需要一个用户空间的监控进程周期性地保存这个数据。
packets	网络接口收到或者发送的数据包个数。
errors	发生错误的次数。
dropped	由于系统资源限制，而丢弃数据包的数量。
overrun	由于发生堵塞，收到的数据包被丢弃的数量。如果接口发生堵塞，就意味着内核或者你的机器太慢，无法处理收到的数据。
mcast	收到的多播数据包数量，只有很少的设备支持这个选项。
carrier	连接介质出现故障的次数，例如：网线接触不好。
collsns	以太网类型介质发生冲突的事件次数。
compressed	压缩数据包的总数。这个指标只适用于使用 VJ 头压缩的网络接口。

如果-s 选项出现两次或者更多次，ip 会输出更为详细的错误信息统计。

```
kuznet@alisa:~ $ ip -s -s link ls eth0

3: eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    2449949362 2786187 0       0       0       0
    RX errors: length  crc     frame   fifo    missed
               0       0       0       0       0
    TX: bytes  packets errors  dropped carrier collsns
    178558497  1783946 332     0       332     35172
    TX errors: aborted fifo    window  heartbeat
               0       0       0       332
kuznet@alisa:~ $
```

这些错误的名字是纯以太化的，对于其它种类的设备，这些域可能有不同的解释。

5.ip address--协议地址管理

缩写	address、addr、a
对象	这里的地址是绑定到网络设备上的协议(IP 或者 IPv6)地址。每个网络设备至少应该有一个协议地址。而且，一个网络设备可以绑定多个

	协议地址。 ip addr 命令能够显示网络设备的协议地址及其性质，添加新的地址，删除旧的地址。
命令	add、delete、flush 和 show(或者 list)

5.1.ip address add--添加一个新的协议地址

缩写：add、a

参数

dev NAME	被操作的设备名
local ADDRESS(default)	接口的地址，地址格式和协议有关。IPv4 地址使用.进行分隔，而 IPv6 地址使用冒号分隔。ADDRESS 可以跟着一个斜杠和表示掩码位数的十进制数字。
peer ADDRESS	点对点接口对端的地址。ADDRESS 也可以跟着一个斜杠和表示掩码位数的十进制数字。
broadcast ADDRESS	接口的广播地址。为了方便，可以使用+和-(注 1)代替广播地址。例如： ip addr add local 192.168.1.1/24 brd + dev eth0 ip addr add local 192.168.1.1/28 brd - dev eth0
label NAME	为每个地址设置一个字符串作为标签。为了和 Linux-2.0 的网络别名兼容，这个字符串必须以设备名开头，接着一个冒号，例如： #ip addr add local 192.168.4.1/28 brd + **label eth0:1** dev eth0
scope SCOPE_VALUE(注 2)	设置地址的有效范围，它用于内核为数据包设置源地址。有效的范围在/etc/iproute2/tr_scopes 文件列出，系统预先设定了一些范围值： • global 这个地址全局有效。 • site 这个地址是局部连接，也就是只有目标地址是这个设备地址时，才有效。 • site (只适用于 IPv6)地址在站点内部有效。 • host 地址在主机内部有效。

*注 1：使用-，ip addr ls 显示的是网络地址；使用+，ip addr ls 显示的是广播地址。

*注 2：有关 scope，在附录 A 中有更为详细的解释。

示例

在回环设备上添加一个回环地址：

```
#ip addr add 127.0.0.1/8 dev lo brd + scope host
```

在以太网接口 eth0 上增加一个地址 10.0.0.1，掩码长度为 24 位(155.155.155.0)，标准广播地址，标签为 eth0:Alias：

```
#ip addr add 10.0.0.1/24 brd + dev eth0 label eth0:Alias
```

5.2.ip address delete--删除一个协议地址

缩写：delete、del、d

参数

这个命令的参数和 ip addr add 命令的参数一致。其中，只有设备名是必需的参数，其它都是可选的。如果没有给定除设备名之外的其它参数，ip 就会删除这个设备的第一个地址。

示例

删除回环设备的一个回环地址。不过，最好不要作这种尝试。

```
#ip addr del 127.0.0.1/8 dev lo
```

以下 shell 代码可以取消设备上的所有 IP 地址。

```
while ip -f inet add del dev eth0;do
   :nothing
done
```

另外，可以使用 ip addr flush 命令取消 IP 地址

5.3.ip address show--显示协议地址

缩写：show、list、lst、sh、ls、l

参数

dev NAME(default)	设备的名字
scope SCOPE_VAL	只列出这个范围的地址
to PREFIX	只列出和 PREFIX 匹配的地址，例如： ip addr ls to 192.168.1.1
label PATTERN	只列出标签匹配 PATTERN 的地址，PATTERN 是一个 shell 风格的正则表达式。
dynamic 和 permanent	这两个参数只适用于 IPv6。使用 dynamic，ip 就只列出动态地址；使用 permanent，ip 就只列出固定地址。

tentative	这个参数只适用于 IPv6，只列出没有通过重复地址检测[参考 2]的地址。
deprecated	这个参数只适用于 IPv6，只列出 deprecated[参考 2]地址。
primary 和 secondary	只列出主(primary)或从(secondary)地址。

输出格式

```
duznet@alisa:~ $ ip addr ls eth0

3: eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff
    inet 193.233.7.90/24 brd 193.233.7.255 scope global eth0
    inet6 3ffe:2400:0:1:2a0:ccff:fe66:1878/64 scope global
dynamic
       valid_lft forever preferred_lft 604746sec
    inet6 fe80::2a0:ccff:fe66:1878/10 scope link
duznet@alisa:~ $
```

输出的头两行和 ip link ls 的输出是相同的。

接着是 IP 和 IPv6 地址、广播地址以及其它的地址属性：范围(scope)、标志(flag)和标签(label)。地址标志由内核设置，系统管理员不能修改。目前，内核定义了以下标志：

secondary	为输出的数据包选择默认源地址时，内核不使用这个地址。如果一个设备已经有了一个地址，又给它设置了同一网段的不同地址，第二个地址就成为从(secondary)地址。例如：eth0 已经有一个地址 192.168.1.108/24，如果又给它一个地址 192.168.1.3/24，192.168.1.3/24 的就会被内核标记为从地址。
dynamic	这个地址是通过无状态的自动配置建立的(stateless autoconfiguration)[参考 2]。如果地址仍然有效，在输出中还包括两个时间信息。preferred_lft 期满后，地址就会变成 deprecated 状态；valiid_lft 期满后，地址将失效。
deprecated	这个地址是不允许的，也就是说，地址虽然有效，但是不能使用它建立新的连接。
tentative	由于重复地址监测[参考 2]还没有完成或者监测失败，这个地址不能使用。

5.4.ip address flush--清除协议地址

缩写：flush、f

简介

这个命令可以清除按照某种条件选择的协议地址。

参数

这个命令的参数和ip address show相同。唯一的区别是，如果不给定参数它什么都不会做。

警告

这个命令(和后面讨论的所有flush命令)非常危险。如果出现错误，将无法恢复，它会清除被操作的地址。

statistics选项

如果在ip addr flush命令中使用了-statistics选项，命令将输出更为详尽的信息。输出的信息包括删除地址的数目和清理地址列表的圈数。如果使用了两次-s选项，ip addr flush会按照上节叙述的格式输出所有被删除的地址。

示例

删除属于私网10.0.0.0/8的所有地址：

```
netadm@amber:~ # ip -s -s a f to 10/8
2: dummy    inet 10.7.7.7/16 brd 10.7.255.255 scope
global dummy
3: eth0    inet 10.10.7.7/16 brd 10.10.255.255 scope
global eth0
4: eth1    inet 10.8.7.7/16 brd 10.8.255.255 scope global
eth1

*** Round 1,deleting 3 addresses ***
*** Flush is complete after 1 round ***
netadm@amber:~ #
```

取消所有以太网卡的IP地址

```
netadm@amber:~ # ip -4 addr flush label "eth0"
```

最后一个例子是对IPv6地址的操作。在启动了转发或者关闭了自动配置之后，你需要取消通过无状态地址自动配置获得的主机地址：

```
netadm@amber:~ # ip -6 addr flush dynamic
```

6.ip neighbour--neighbour/arp表管理命令

缩写	neighbour、neighbor、neigh、n
对象	邻接(neighbour)对象实现同一网段协议地址和链路层地址的绑定。在内核中，这些条目被组织到表中。IPv4 的相邻表也被叫做 ARP 表。ip neighbour 命令支持对条目及其属性的显示、添加和删除。
命令	add、change、replace、delete、fulsh、show(或者 list)

附录 B 将详细描述如何使用 ip 管理代理 ARP/NDISC。

6.1.ip neighbour add -- 添加一个新的邻接条目
ip neighbour change--修改一个现有的条目
ip neighbour replace--替换一个已有的条目

缩写：add、a；change、chg；replace、repl
简介：这三个命令用来建立一个邻接表的条目或者更新现有的邻接表条目。
参数

to ADDRESS(default)	相邻的协议地址。可以是 IPv4 或者 IPv6。
dev NAME	和相邻节点连接的设备。
lladdr LLADDRESS	邻居的链路层地址。LLADDRESS 可以为空。
nud NUD_STATE	邻接条目的状态。**nud** 是 *Neighbour Unreachability Detection* 的缩写。可能的状态包括： *permanent*--邻接条目永远有效并且只能由管理员删除。 *noarp*--邻接条目有效，不必对其有效性进行确认。在其生命期期满时会被删除。 *reachable*--在超时时间之内，这个邻接条目是有效的。 *stale*--这个邻接条目是有效的，但是比较可疑。如果条目是有效的，*ip neigh* 不会改变邻接状态，也不会修改其地址。

示例

在设备 eth0 上，为地址 10.0.0.3 添加一个 permanent ARP 条目：

```
ip neigh add 10.0.0.3 lladdr 0:0:0:0:0:1 dev eth0 nud
perm
```

把状态改为 reachable

```
ip neigh chg 10.0.0.3 dev eth0 nud reachable
```

6.2.ip neighbour delete--删除一个邻接条目

缩写：delete、del、d
简介

这个命令用来删除一个邻接条目

参数

这个命令的参数和ip neigh add命令的相同，只不过lladdr和nud将被忽略。

示例

删除设备eth0上的一个ARP条目10.0.0.3

```
ip neigh del 10.0.0.3 dev eth0
```

执行了删除命令之后，被删除的条目不会马上消失，它会在系统的下次垃圾收集时被删除。如果被操作的条目正在使用，将不能被删除。

警告

如果试图删除或者手工修改一个由内核建立的*noarp*条目，会导致一些不可预知的行为。

6.3.ip neighbour show--显示网络邻居的信息

缩写：show、list、sh、ls
简介

这个命令用于显示网络邻居信息。

参数

to ADDRESS(default)	指定要显示的地址
dev NAME	只显示设备NAME的邻居
unused	只显示当前没有使用的邻居
nud NUD_STATE	只列出处于NUD_STATE状态的邻接条目。NUD_STATE的值下面将会介绍。*nud all*表示所有的状态。这个选项可以使用多次。如果缺少这个选项，*ip*会列出除*none*和*noarp*状态的所有条目。

输出格式

```
duznet@alisa:~ $ ip neigh ls

:: dev lo lladdr 00:00:00:00:00:00 nud noarp
fe80::200:cff:fe76:3f85 dev eth0 lladdr 00:00:0c:76:3f:86
router
    nud stale
0.0.0.0 dev lo lladdr 00:00:00:00:00:00 nud noarp
193.233.7.254 dev eth0 lladdr 00:00:0c:76:3f:85 nud reachable
193.233.7.85 dev eth0 lladdr 00:e0:1e:63:39:00 nud stale
kuznet@alisa:~ $
```

每行的第一部分是网络邻居的协议地址。第二部分是设备名。省下的部分是这个邻接条目的信息。

lladdr 是这个设备的链路层地址。

nud 是条目代表连接的状态。下面是状态的完整列表和简单描述：

none	网络邻居的状态为空。
incomplete	这个邻居正在被解析。
reachable	网络邻居有效并且可达。
stale	邻居有效，但是可能不可达。因此，内核将在首次传输时进行检查。
delay	一个数据包已经发到处于 *stale* 的网络邻居,内核在等待应答信息。
probe	*delay* 计时器过期，还没有收到确认信息。内核开始使用 ARP/NDISC 消息包探测这个网络邻居。
failed	解析失败。
noarp	网络邻居有效，不必检查。
permanent	这是一个 *noarp* 条目，只有系统管理员可以从邻接表中把它删除。

在这些状态中，除了 *none*、*faliled* 和 *incomplete*.

IPVv6 网络邻居可以有一个叫做 *router* 的标志，它表示这个节点是一个 IPv6 路由器。

-statistics

-statistics 选项可以显示很多有用的信息。例如：

```
kuznet@alisa:~ $ ip -s n ls 193.233.7.254

193.233.7.254. dev eth0 lladdr 00:00:0c:76:3f:85 ref 5
    used 12/13/20 nud reachable
kuznet@alisa:~ $
```

输出信息里面多了 ref 和用斜缸分开的三个时间。ref 表示有多少用户使用这个条目；三个时间分别是使用时间、确认时间和刷新时间。因此，上面输出中的时间表示：

```
    条目 12 秒之前刚刚使用过；

    13 秒之前被确认；
    20 秒之前被更新。
```

6.4.ip neighbour flush--清除邻接条目

缩写：flush、f
简介

这个命令用来清除符合某个条件的邻接表条目。

参数

这个命令的参数和 ip neigh sh 相同。不同之处是，如果没有参数，它什么也不会做。而且，默认情况下，被删除的条目不包括处于 permanent 和 noarp 状态的条目。

-statistics

使用了-statistics 选项，这个命令的输出将更为详尽。它会输出删除的条目数和清除邻接表遍历的次数。如果使用了两个-s 选项，命令的输出将包括被删除条目的信息。

示例

```
netadm@alisa:~ # ip -s -s n f 193.233.7.254

193.233.7.254 dev eth0 lladdr 00:00:0c:76:3f:85 ref 5
    used 12/13/20 nud reachable

*** Round 1,deleting 1 entries ***
*** Flush is complete after 1 round ***
netadm@alisa:~ #
```

7.路由表管理

7.1.缩写

route、ro、r

7.2.对象

路由条目保存在内核的路由表中，它们包含寻找到其它网络节点的路径信息。路由表条目都包括一对网络地址/掩码长度以及可选的TOS值等信息。如果数据包目的地址位于属于路由条目的的范围，以及路由的TOS等于0或者等于数据包的TOS，它就匹配路由条目。如果一个数据包匹配多个路由条目，系统内核将按照以下规则决定选择哪个路由：

注：作者在文中把**地址被子网掩码屏蔽后的部分/掩码长度**这种表达方式叫做前缀（prefix）。例如：10/8表示网络10.0.0.0，子网掩码长度是8位；10.1/16表示网络10.1.0.0，子网掩码长度是16位；

范围最小的优先匹配，较大的放弃；

路由TOS等于数据包TOS的匹配，不等于的放弃；
如果经过上面两步的选择，还有数个路由，就选择优先值最高的路由；
如果还有数个路由可供选择，就重复进行第一步。

为了简化，我们使用{prefix,tos,preference}来标记每个路由。

7.3.路由属性

路由条目提供IP数据包投递所需的路由信息、数据(例如：输出设备、下一跳的路由器)和一些可选属性(例如：路径的最大传输单元MTU或者源地址等)。这些属性将在后面的章节详细介绍。

7.4.路由类型

路由的设置以及其它的可选属性都依赖于路由类型。最重要的路由类型是*unicast*路由，这种类型的路由表示到另外主机的真实路由。一般情况下，通常的路由表只有这种类型的路由条目。不过，还存在其它类型的路由，使用的语法也不相同。Linux-2.2理解以下几种类型的路由：

unicast	这种类型的路由描述到目的地址的真实路径。
unreachable	这些目的地址是不可达的。如果发过去的数据包都被丢弃并

	且收到 ICMP 信息 *host unreachable*，目的地址就会被标记为不可达。在这种情况下，本地发送者将返回 EHOSTUNREACH 错误。
blackhole	这些目的地址不可达，而且发过去的数据包都被丢弃。在这种情况下，本地发送者将返回 EINVAL 错误。
prohibit	这些路由是不可达的。发过去的数据包都被丢弃，而且产生 ICMP 信息 *communication administratively prohibited* 。本地发送者会返回 EACCESS 错误。
local	目的地址被分配给本机。数据包通过回环被投递到本地。
broadcast	目的地址是广播地址，数据包作为链路广播发送。
throw	和策略规则(policy rule)一块使用的控制路由。如果选择了这种路由，就会认为没有发现路由，在这个表中的查询就会被终止。没有找到策略路由就相当于在路由表中没有找到路由，数据包会被丢弃，并产生 ICMP 信息 *net unreachable*。本地发送者会返回 ENETUNREACH 错误。
nat	特定的 NAT 路由。目标地址属于哑地址（或者称为外部地址），在转发前需要进行地址转换。
anycast	目标是 *anycast* 地址，被分配给本机。这类地址和本地地址大同小异，不同的是这类地址不能用于任何数据包的源地址。
multicast	使用多播路由。在普通的路由表中，这种路由并不存在。

7.5.路由表

从 Linux-2.2 开始，内核把路由归纳到许多路由表中，这些表都进行了编号，编号数字的范围是 1 到 255。另外，为了方便，还可以在 */etc/iproute2/rt_tables* 中为路由表命名。默认情况下，所有的路由都会被插入到表 *main*(编号 254)中。在进行路由查询时，内核只使用路由表 *main*。

实际上，还有另外一个路由表也一直存在，这个表是不可见的，而且极为重要。这就是表 *local*。这个表保存本地和广播路由。内核会自动维护这个路由表，通常系统管理员没有必要对它进行修改，甚至不必看到。

在使用策略路由（policy routing）时，我们将使用多个路由。在这种情况下，表识别符有很多参数，因此需要使用 *{prefix,tos,preference}* 的形式唯一地识别每个路由。

7.6.ip route add -- 添加新路由
ip route change -- 修改路由
ip route replace -- 替换已有的路由

缩写：add、a；change、chg；replace、repl

参数

to PREFIX 或者 to TYPE PREFIX(default)	路由的目标前缀(prefix)。如果 *TYPE* 被忽略，ip 命令就会使用默认的类型 *unicast*。其它的类型在上一节都有介绍。PREFIX 是一个 IP 或者 IPv6 地址，也可以跟着一个斜杠和掩码长度。如果没有掩码长度，ip 命令就假定是一个单一 ip 地址。另外，还有一个特殊的 PREFIX--default（缺省路由），它等于 IPv4 的 0/0，或者 IPv6 的 *::/0*。
tos TOS 或者 defield TOS	定义服务类型关键词。在进行路由匹配时，内核首先比较数据包的 TOS 和 route 的 TOS，如果没有和数据包 TOS 相同的路由，还可以选择 TOS 等于 0 的路由。TOS 或者是一个十六进制的数字，或者是一个由 /etc/iproute2/rt_dsfield 文件定义的识别符。
metric NUMBER 或者 preference NUMBER	定义路由的优先值，NUMBER 时一个任意的 32 位数字。
table TABLEID	路由要加入的表。TABLEID 或者是一个数字或者是 /etc/iproute2/rt_tables 文件定义的一个字符串。如果没有这个参数，ip 命令就会把路由加入到表 *main* 中，本地(local)、广播(broadcast)和网络地址转换(nat)路由除外。在默认情况下，这些类型的路由都会被加入表 *local* 中。
dev NAME	输出设备的名字
via ADDRESS	指定下一跳路由器的地址。实际上，这个域的可靠性取决于路由类型。对于通常的 *unicast* 路由，它或者是真正的下一跳路由器地址，或者如果它是 BSD 兼容模式安装的直接路由，它可以是一个网络接口的本地地址。对于 NAT 路由，它是转换后的地址。
src ADDRESS	在向目的 prefix 发送数据包时选择的源地址。
realm REALMID	指定路由分配的 realm。REALM 可以是一个数字或者 /etc/iproute2/rt_realms 文件定义的一个字符串。有关 realm 更为详细的信息请看附录（Route realms and policy propagation, rtacct）。
mtu MTU 或者 mtu lock MTU	设置到达目的路径的最大传输单元(MTU)。如果没有使用修饰符 *lock*，内核会通过路径最大传输单元发现（Path MTU Discovery）机制更新 MTU；如果使用了修饰符 *lock*，内核就不会测试路径的最大传输单元。在这种情况下，发出的所有 IPv4 数据包 DF 域都会被设置为 0（允许分片），对于 IPv6 数据包也允许分片。
window NUMBER	设置到目的地址 TCP 连接的最大窗口值，以字节为

	单位。使用这个参数可以限制对端发送数据的速率。
rtt NUMBER	估算初始往返时间（Round Trip Time）
rttvar NUMBER	估算初始往返时间偏差(RTT variance)
ssthresh NUMBER	估算慢启动阀值(slow start threshould)
cwnd NUMBER	把拥挤窗口(congestion window)值锁定为NUMBER。如果没有lock标记，这个值会被忽略。
advmss NUMBER	设置在建立TCP连接时，向目的地址声明的最大报文段大小(Maximal Segment Size,MSS)。如果没有设置，Linux内核会使用计算第一跳的最大传输单元得到的数值。
nexthop NEXTHOP	设置多路径路由的下一跳地址。NEXTHOP比较复杂，它的语法和以下高层参数类似： *via ADDRESS*--表示下一跳路由器； *dev NAME*--表示输出设备； *weight NUMBER*--在多路由路径中，这个元素的权重。表示相对带宽或者服务质量。
scope SCOPE_VAL	路由前缀(prefix)覆盖的范围。SCOPE_VAL可以是一个数字，也可以是/etc/iproute2/rt_scope文件定义的一个字符串。如果没有这个参数，ip命令就会根据具体情况猜测：对于经过网关的*unicast*路由，就设置为global；对于直连的*unicast*路由和广播路由，就设置为link；对于本地路由，就设置为host。
protocol RTPROTO	本条路由得路由协议识别符。RTPROTO可以是一个数字，也可以是/etc/iproute2/rt_protos文件定义的一个字符串。如果使用时没有提供这个参数，ip命令就使用默认值*boot*(也就是说，ip命令认为添加路由的人不知道自己做了些什么)。有些协议值有其固定的解释： *redirect*--路由是由ICMP重定向加入的； *kernel*--路由是由内核在自动配置期间加入的； *boot*--路由是启动过程中加入的。如果一个路由监控程序将要启动，这些路由都会被清除； *static*--为了覆盖动态路由，由系统管理员手工添加的路由。路由监控程序也会优先考虑这类路由，甚至可能通告给其对端； *ra*--路由是通过路由发现协议加入的(Router Discovery Protocol)。 其它的值没有保留，系统管理员可以自由分配(或者

	不分配)给协议标记。至少，路由监控程序应该注意对一些唯一协议值的设置，这些协议值在 rtnetlink.h 文件或者 rt_protos 数据库中分配。
onlink	假装和下一跳路由器是直接相连的，即使它没有匹配任何接口前缀(prefix)。
equalize	允许把数据包随机从多个路由发出。如果没有这个路由修饰符，内核就会冻结下一跳路由的地址。

示例

设置到网络 10.0.0/24 的路由经过网关 193.233.7.65

```
ip route add 10.0.0/24 via 193.233.7.65
```

修改到网络 10.0.0/24 的直接路由，使其经过设备 *dummy*

```
ip route chg 10.0.0/24 dev dummy
```

加入缺省多路径路由，让 ppp0 和 ppp1 分担负载(注意：scope 值并非必需，它只不过是告诉内核，这个路由要经过网关而不是直连的。实际上，如果你知道远程端点的地址，使用 *via* 参数来设置就更好了)。

```
ip route add default scope global nexthop dev ppp0
                                  nexthop dev ppp1
```

设置 NAT 路由。在转发来自 192.203.80.144 的数据包之前，先进行网络地址转换，把这个地址转换为 193.233.7.83（回来的转换将会在后面的章节路由策略中介绍）。

```
ip route add nat 192.203.80.142 via 193.233.7.83
```

7.7.ip route delete-- 删除路由

缩写：delete、del、d

参数

ip route del 使用和 *ip route add* 相同的参数，不过语法稍有不同。这个命令使用关键词（to、tos、preference 和 table）选择要删除的路由。如果命令中使用了可选属性，ip 命令会校验这个属性和要删除的路由是否一致；如果没有给定关键词或者属性不一致，*ip route del* 会执行失败。

示例

删除上一节命令加入的多路径路由

```
ip route del default scope global nexthop dev ppp0
                                  nexthop dev ppp1
```

7.8.ip route show -- 列出路由

缩写：show、list、sh、ls、l

简介

使用这个命令，你可以看到路由表的内容，或者查询符合某些条件的路由。

参数

to SELECTOR(default)	只选择到给定地址的路由。 SELECTOR 由修饰符(root、match、exact，可选)和一个前缀(prefix)组成。*root PREFIX*表示选择前缀(prefix)不短于 PREFIX 的路由，例如：root 0/0 将选在路由表里面的全部路由；*match PREFIX*表示选择前缀(prefix)不长于 PREFIX 的路由，match 10.1/16 会选择前缀(prefix)是 10.1/16、10./8 和 0/0 的全部路由；而 *exact PREFIX(或者 just PREFIX)*表示精确匹配。如果没有这些选项(ip route ls)，*ip 命令*就假定是 *ip route ls to root 0/0*，将列出系统的所有路由。
tos TOS 或者 dsfield TOS	只列出 tos 等于 TOS 的路由
table TABLEID	列出路由表 TABLEID 里面的路由。缺省设置是 *table main*。TABLEID 或者是一个真正的路由表 ID 或者是/etc/iproute2/rt_tables 文件定义的字符串，或者是以下的特殊值： *all* -- 列出所有表的路由； *cache* -- 列出路由缓存的内容。
cloned 或者 cached	列出被克隆出来的路由（由于某些路由属性改变，例如：MTU，而由某些路由派生出来的路由）。实际上，它的内容和表缓存的内容是一样的。
from SELECTOR	和 *to 的语法是相同的，只不过由目的地址换为源地址而已。**注意**：这个选项之适用于被克隆出来的路由。*
protocol RTPROTO	只列出协议是 RTPROTO 的路由
scope SCOPE_VAL	只列出范围是 SCOPE_VAL 的路由
type TYPE	只列出类型为 TYPE 的路由
dev NAME	只列出通过设备 NAME 的路由
via PREFIX	只列出下一跳通过 PREFIX 的路由
src PREFIX	只列出源地址属于 PREFIX 的路由
realm REALMID 或者 raalm FROMREALM/TOREALM	只列出 realm 为 REALMID 的路由

示例

```
    计算使用 gated/bgp 协议的路由个数

      kuznet@amber:~ $ ip route ls proto gated/bgp |wc
         1413    9891    79010
      kuznet@amber:~ $
    计算路由缓存里面的条数，由于被缓存路由的属性可能大于一行，
以此需要使用-o选项
      uznet@amber:~ $ ip -o route ls cloned |wc
         159    2543    18707
      kuznet@amber:~ $
```

输出格式

通常，在这个命令输出的信息中，每个路由纪录占一行。不过，有时某些纪录可能会超过一行，例如被克隆出来的路由或者包含一些额外的信息。如果在命令中使用了*-o*选项，在每个纪录中，会使用代替回车作为回行标记。例如：

```
kuznet@amber:~ $ ip ro ls 193.233.7/24

193.233.7.0/24 dev eth0 proto gated/conn scope link src
193.233.7.65 realms inr.ac
kuznet@amber:~ $
```

如果是列出被克隆的条目，输出信息将是另外的形式。例如：

```
kuznet@amber:~ $ ip ro ls 193.233.7.82 tab cache

193.233.7.82 from 193.233.7.82 dev eth0 src 193.233.7.65
realms inr.ac/inr.ac
    cache <src-direct,redirect>  mtu 1500 rtt 300 iif eth0
193.233.7.82 dev eth0  src 193.233.7.65 realms inr.ac
    cache  mtu 1500 rtt 300
kuznet@amber:~ $
```

输出信息的第二行是以关键词 *cache* 开头的，显示路由的其它缓存标记和属性。本行的第一个域是 *cache <缓存标记>*，缓存标记包括：

local	数据包被投递到本地。它适用于本地回环单向传播(unicast)路由，如果这个主机是对应广播组的一个成员，它也适用于广播路由何多播路由。
reject	路径无效。任何试图通过这个路由的企图都会导致错误。
mc	目的是多播地址(multicast)。

brd	目的是广播地址(broadcast)。
src-direct	源地址是在一个直接连接的接口。
redirected	路由是由 ICMP 重定向建立的。
redirect	数据包通过这个路由将触发 ICMP 重定向。
fastroute	路由适合用于快速路由(fastroute)。
equalize	使数据包随机地通过这个路由。
dst-nat	目的地址需要进行地址转换。
src-nat	源地址需要进行地址转换。
masq	源地址需要伪装(masquerading)。
notify	修改/删除这个路由将触发 RTNETLINK 报警。

接着是一些路由属性，支持的属性如下：

error	对于 *reject* 路由，这是返回给本地发送者的错误码。这些错误码也会被转换为 ICMP 错误码，发送给远程发送者。
expires	到了超时时间，这个条目就会消失。
iif	需要这个路由的数据包如期到达这接口。

统计选项

如果在命令中使用*-statistics*选项，ip 命令会给出一些更为详尽的信息：

users	使用这个路由的用户数。
age	显示这个路由最后使用时的时间。
used	自从建立这个路由以来，它被查询的次数。

7.9.ip route flush -- 擦除路由表

缩写：flush、f
简介

使用这个命令，可以很方便地删除符合某些条件的路由。

参数

这个命令的参数和 *ip route show* 命令的参数相同，只不过被操作的路由表不会被显示出来。它和 *ip route show* 命令唯一的区别是它们的缺省操作，*ip route show* 会显示出路由表 *main* 的所有条目，而 *ip route flush* 只会给出帮

助信息，不对路由表进行任何操作。至于这个区别的原因，恐怕不必多做解释了吧？

统计选项

如果在这个命令中使用了*-statistics*选项，它就会显示一些冗余信息。这些信息包括：被删除的路由数和删除过程中遍历路由表的次数。如果这个选项使用了两次，ip 还会输出被删除路由的详细信息。

示例

第一个例子是删除路由表*main*中的所有网关路由（例如：在路由监控程序挂掉之后）：

```
netadm@amber:~ # ip -4 ro flush scope global type unicast
```

第二个例子是清除所有被克隆出来的 IPv6 路由：

```
netadm@amber:~ # ip -6 -s -s ro flush cache

3ffe:2400::220:afff:fef4:c5d1 via
3ffe:2400::220:afff:fef4:c5d1
  dev eth0  metric 0
    cache  used 2 age 12sec mtu 1500 rtt 300
3ffe:2400::280:adff:feb7:8034 via
3ffe:2400::280:adff:feb7:8034
  dev eth0  metric 0
    cache  used 2 age 15sec mtu 1500 rtt 300
3ffe:2400::280:c8ff:fe59:5bcc via
3ffe:2400::280:c8ff:fe59:5bcc
  dev eth0  metric 0
    cache  users 1 used 1 age 23sec mtu 1500 rtt 300
3ffe:2400:0:1:2a0:ccff:fe66:1878 via
3ffe:2400:0:1:2a0:ccff:fe66:1878
  dev eth1  metric 0
    cache  used 2 age 20sec mtu 1500 rtt 300
3ffe:2400:0:1:a00:20ff:fe71:fb30 via
3ffe:2400:0:1:a00:20ff:fe71:fb30
  dev eth1  metric 0
    cache  used 2 age 33sec mtu 1500 rtt 300
ff02::1 via ff02::1 dev eth1  metric 0
    cache  users 1 used 1 age 45sec mtu 1500 rtt 300
```

```
*** Round 1, deleting 6 entries ***
*** Flush is complete after 1 round ***
netadm@amber:~ # ip -6 -s -s ro flush cache
Nothing to flush.
netadm@amber:~ #
```

第三个例子是在*gated*程序挂掉之后，清除所有的BGP路由：

```
netadm@amber:~ # ip ro ls proto gated/bgp |wc

   1408   9856   78730
netadm@amber:~ # ip -s ro f proto gated/bgp
*** Round 1, deleting 1408 entries ***
*** Flush is complete after 1 round ***
netadm@amber:~ # ip ro f proto gated/bgp
Nothing to flush.
netadm@amber:~ # ip ro ls proto gated/bgp
netadm@amber:~ #
```

7.10.ip route get -- 获得单个路由

缩写：get、g
简介

使用这个命令可以获得到达目的地址的一个路由以及它的确切内容。

参数

to ADDRESS(default)	目标地址
from ADDRESS	源地址
tos TOS或者 dsfield TOS	服务类型
iif NAME	数据包进来的设备
oif NAME	数据包出去的设备
connected	*ip route get*命令至少要有参数*to ADDRESS*。使用*connected*参数，如果没有给出源地址(*from ADDRESS*)，ip就会重新在路由表中查询能够到达目的地址的源地址，给出获得的第一个源地址到目的地址的路由。如果使用了策略路由，会有所不同。

*ip route get*命令和*ip route show*命令执行的操作是不同的。*ip route show*命令只是显示现有的路由，而*ip route get*命令在必要时会派生出新的路由。

输出格式

这个命令的输出格式和*ip route ls*相同。

示例

搜索到193.233.7.82的路由

```
kuznet@amber:~ $ ip route get 193.233.7.82
193.233.7.82 dev eth0  src 193.233.7.65 realms inr.ac
    cache  mtu 1500 rtt 300
kuznet@amber:~ $
```

搜索目的地址是193.233.7.82，来自193.233.7.82，从eth0设备到达的路由（这条命令会产生一条非常有意思的路由，这是一条到193.233.7.82的回环路由）

```
kuznet@amber:~ $ ip r g 193.233.7.82 from 193.233.7.82
iif eth0
193.233.7.82 from 193.233.7.82 dev eth0  src
193.233.7.65
  realms inr.ac/inr.ac
    cache <src-direct,redirect>  mtu 1500 rtt 300 iif
eth0
kuznet@amber:~ $
```

获得一个多播路由，数据包来自主机193.233.7.82，从eth0设备进入，目的地址是多播组地址224.2.127.254（需要运行多播路由监控程序pimd）。这个命令产生的路由与上面的不大相同，它包含常规部分和多播部分。常规部分用于把数据包投递到本地ip监控程序。这里，本地地址不是多播组的成员，因此这个路由没有local标记，只用于转发数据包。这个路由的输出设备是回环设备。多播部分包含额外的输出接口。

```
kuznet@amber:~ $ ip r g 224.2.127.254 from 193.233.7.82
iif eth0
multicast 224.2.127.254 from 193.233.7.82 dev lo
  src 193.233.7.65 realms inr.ac/cosmos
    cache <mc> iif eth0 Oifs: eth1 pimreg
kuznet@amber:~ $
```

下面我们举一个复杂一些的例子。我们首先为一个目标地址添加一个无效的网关路由，而实际上和这个地址是直连的。

```
netadm@alisa:~ # ip route add 193.233.7.98 via 193.233.7.254

netadm@alisa:~ # ip route get 193.233.7.98
193.233.7.98 via 193.233.7.254 dev eth0  src 193.233.7.90
    cache  mtu 1500 rtt 3072
netadm@alisa:~ #
```

然后，我们ping一下193.233.7.98:

```
netadm@alisa:~ # ping -n 193.233.7.98

PING 193.233.7.98 (193.233.7.98) from 193.233.7.90 : 56 data
bytes
From 193.233.7.254: Redirect Host(New nexthop: 193.233.7.98)
64 bytes from 193.233.7.98: icmp_seq=0 ttl=255 time=3.5 ms
From 193.233.7.254: Redirect Host(New nexthop: 193.233.7.98)
64 bytes from 193.233.7.98: icmp_seq=1 ttl=255 time=2.2 ms
64 bytes from 193.233.7.98: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 193.233.7.98: icmp_seq=3 ttl=255 time=0.4 ms
64 bytes from 193.233.7.98: icmp_seq=4 ttl=255 time=0.4 ms
^C
--- 193.233.7.98 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.4/1.3/3.5 ms
netadm@alisa:~ #
```

输出结果可以看出，路由器193.233.7.254知道有更好的路由，因此送回一个ICMP重定向信息。然后，我们再看看路由表的情况：

```
netadm@alisa:~ # ip route get 193.233.7.98

193.233.7.98 dev eth0  src 193.233.7.90
    cache <redirected>  mtu 1500 rtt 3072
netadm@alisa:~ #
```

8.ip rule -- 路由策略数据库管理命令

8.1.缩写

rule、ru

8.2.对象

路由策略数据库的规则用于控制选择路由的算法。

Internet 上采用的路由算法一般是基于数据包目的地址的。理论上，也可以由 TOS 域决定，不过这没有实际应用。要了解经典路由算法的详细情况请参考 *RFC-1812*。

而在某些情况下，我们不只是需要通过数据包的目的地址决定路由，可能还需要通过其他一些域：源地址、IP 协议、传输层端口甚至数据包的负载。这就叫做：*策略路由*(policy routing)。

注意：策略路由(*policy routing*)不等于路由策略(*rouing policy*)。

在这种情况下，传统的基于目的地址的路由表就无法满足要求了，需要使用路由策略数据库(*routing policy database,RPDB*)代替，通过它选择执行某些路由。这些规则可以由很多不同的状态，而且它们没有天生的次序，要由系统管理员决定。RPDB 可以匹配以下的域:

```
数据包的源地址;

数据包的目的地址;
服务类型(Type of Service);
进入的网络接口;
```

匹配 IP 协议和传输层端口也是可能的，不过这要依靠 *iptables* 或者 *ipchains* 通过 *fwmark* 为某些数据包做标记，并重定向。

每个路由策略由一个*选择符(selector)*和一个*操作(action)*组成。系统按照顺序搜索路由策略数据库，把选择符和*{源地址、目的地址、进入接口、tos、fwmark}*等关键词进行匹配，如果匹配成功，就执行 *action* 定义的操作。操作或者成功返回，或者失败并且中止对路由策略。否则，系统继续查询路由策略数据库。

操作如何定义？最原始的操作是选择*下一跳(nexthop)和输出设备(output device)*。Cisco IOS 使用这种方式，我们姑且把这叫做*匹配并设置(match & set)*。而 Linux 的方式则更为灵活，Linux 允许的操作包括：基于目的地址的路由表查询以及按照最长匹配的原则从路由表中选择路由。因此，*匹配并设置(match & set)*的方式只是一个最简单的特例而已。

再系统启动时，内核会为路由策略数据库配置三条缺省的规则：

优先级	选择符	操作	解释
0	匹配	查询路由表	路由表 *local* 是一个特殊的路由表，包含对

	任何条件	*local(ID 255)*	于本地和广播地址的高优先级控制路由。rule 0 非常特殊，不能被删除或者覆盖。
32766	匹配任何条件	查询路由表 *main(ID 254)*	路由表 *main(ID 254)*是一个通常的表，包含所有的无策略路由。系统管理员可以删除或者使用另外的规则覆盖这条规则。
32767	匹配任何条件	查询路由表 *default(ID 253)*	路由表 *default(ID 253)*是一个空表，它是为一些后续处理保留的。对于前面的缺省策略没有匹配到的数据包，系统使用这个策略进行处理。这个规则也可以删除。

不要混淆路由表和策略：规则指向路由表，多个规则可以引用一个路由表，而且某些路由表可以没有策略指向它。如果系统管理员删除了指向某个路由表的所有规则，这个表就没有用了，但是仍然存在，直到里面的所有路由都被删除，它才会消失。

8.3.规则类型

路由策略规则数据库可以包括如下类型的规则：

unicast	返回从被引用的路由表中发现的路由
blackhole	丢弃数据包，不做任何反应
unreachable	产生网络不可达(Network is unreachable)的 ICMP 错误信息
prohibit	产生通讯被禁止(Communication is administratively prohibited)的 ICMP 错误信息
nat	把数据报的源地址转换为其它的值。详情请参考附录 C

8.4.命令

add、delete、show(或者 list)

8.5.ip rule add -- 插入新的规则
ip rule delete -- 删除规则

缩写：add、a；delete、del、d

参数

type TYPE(default)	这个规则的类型。有效的类型上一节已经介绍过了。
from PREFIX	匹配的源地址
iif NAME	选择数据包进入的设备。如果接口是回环设备，这个规

	则就只匹配源于本机的数据包。这意味着，你可以为本机发出的数据包和要转发的数据包分别建立路由表，使两者完全隔离。
tos TOS 或者 dsfield TOS	选择匹配的 TOS 值
fwmark MARK	选择要匹配的 *fwmark* 值
priority PREFERENCE	设置这个规则的优先级。每个规则的优先级都应该明确设置为一个唯一的数值。实际上，由于历史的原因，*ip roule add* 命令无需任何优先级的值，也不必是唯一的。如果用户没有在命令中提供优先级的值，内核会自动选择。如果用户提供的优先级值已经存在，内核也不会拒绝这次请求，而是在相同优先级的规则前面插入新的规则。
table TABLEID	如果规则选择符匹配，就被查询的路由表识别符。
realms FROM/TO	如果规则匹配和路由表查询成功，选择的 realms 值。
nat ADDRESS	设置要进行网络地址转换的 IP 地址段。ADDRESS 或者是进行网络地址转换 ip 地址段，或者是一个本机地址，甚至可以是 0。

警告

使用上面两个命令对路由策略数据库进行的任何修改都不会马上生效。只有使用 *ip route flush cach* 命令刷新路由缓存之后才会生效。

示例

通过路由表 inr.ruhep 路由来自源地址为 192.203.80/24 的数据包

```
ip ru add from 192.203.80/24 table inr.ruhep prio 220
```

把源地址为 193.233.7.83 的数据报的源地址转换为 192.203.80.144，并通过表 1 进行路由

```
ip ru add from 193.233.7.83 nat 192.203.80.144 table 1
prio 320
```

删除无用的缺省规则

```
ip ru del prio 32767
```

8.6.ip rule show -- 列出路由规则

缩写：show、list、sh、ls、l

参数

好消息，这个命令没有参数。

输出格式

```
kuznet@amber:~ $ ip ru ls

0:      from all lookup local
200:    from 192.203.80.0/24 to 193.233.7.0/24 lookup main
210:    from 192.203.80.0/24 to 192.203.80.0/24 lookup main
220:    from 192.203.80.0/24 lookup inr.ruhep realms
inr.ruhep/radio-msu
300:    from 193.233.7.83 to 193.233.7.0/24 lookup main
310:    from 193.233.7.83 to 192.203.80.0/24 lookup main
320:    from 193.233.7.83 lookup inr.ruhep map-to
192.203.80.144
32766:  from all lookup main
kuznet@amber:~ $
```

每行第一部分的数字是规则的优先级，接着是选择符。

关键词 *lookup* 后面接着路由表识别符。

如果规则要进行网络地址转换，还需要一个关键词 *map-to* 设置转换以后的地址。

上面的示例非常简单，192.203.80.0/24 和 193.233.7.0/24 组成内部网络，但是它们向外发送数据包要通过不同的路由。主机 193.233.7.83 和外界会话时，地址需要转换为 192.203.80.144。

9.ip maddress -- 多播地址管理

9.1.对象

这个命令管理的对象是多播地址。

9.2.命令

add、delete、show(或者 list)

9.3.ip maddress show -- 列出多播地址

缩写：show、list、sh、ls、l

参数

dev NAME(defautl)	设备名

输出格式

```
kuznet@alisa:~ $ ip maddr ls dummy

2:  dummy
    link  33:33:00:00:00:01
    link  01:00:5e:00:00:01
    inet  224.0.0.1 users 2
    inet6 ff02::1
kuznet@alisa:~ $
```

输出的第一行是设备的索引和设备名。后面几行是多播地址，每行由协议识别符开头。关键词 *link* 表示这是链路层多播地址。

如果一个多播地址有几个用户，那么用户数就在 *users* 关键词之后列出。上面的例子没有出现关键词 *static*，它表示这个地址是由 *ip maddr add* 命令加入的。

9.4.ip maddress add -- 加入多播地址
ip maddress delete -- 删除多播地址

缩写：add、a；delete、del、d

简介

使用这两个命令，我们可以添加 / 删除在网络接口上监听的链路层多播地址。这个命令只能管理链路层地址。

参数

address LLADDRESS(default)	链路层多播地址
dev NAME	加入 / 脱离这个多播地址的设备

示例

让我们继续上一小节的例子

```
netadm@alisa:~ # ip maddr add 33:33:00:00:00:01 dev dummy

netadm@alisa:~ # ip -O maddr ls dummy
2:  dummy
    link  33:33:00:00:00:01 users 2 static
    link  01:00:5e:00:00:01
netadm@alisa:~ # ip maddr del 33:33:00:00:00:01 dev dummy
```

注意：ip 命令和内核都不会检查多播地址的有效性。这意味着你可以使用 unicast 地址代替多播地址。大多数驱动都会忽略 unicast 地址，但是有些驱动 (例如：tulip) 会把这个 unicast 地址加入其过滤器。这样作的效果有些奇怪，

如果你使用了别的主机或者路由器的地址作为多播地址，你就可以收到发送到它们的数据包。不过，这并非一个 bug，而是内核的一个功能。它可以用于网络监视。

10.ip mroute -- 多播路由缓存管理

10.1.缩写

mroute、mr

10.2.对象

这个命令的操作对象是多播路由缓存条目，这个缓存是由一个用户空间的多播路由监控进程(例如 pimd 或者 mrouted)建立的。

目前，由于受和多播路由引擎接口的限制，还不能通过 ip 命令修改多播路由对象，因此我们只能查看。

10.3.命令

show 或者 list

10.4.ip mroute show -- 列出多播路由缓存条目

缩写：show、list、sh、ls、l

参数

to PREFIX(default)	选择到目的多播地址是 PREFIX
iif NAME	接收多播数据包的网络接口
from PREFIX	PREFIX 选择多播路由的 IP 源地址

输出格式

```
kuznet@amber:~ $ ip mroute ls

(193.232.127.6, 224.0.1.39)       Iif: unresolved
(193.232.244.34, 224.0.1.40)      Iif: unresolved
(193.233.7.65, 224.66.66.66)      Iif: eth0       Oifs: pimreg
kuznet@amber:~ $
```

多播路由缓存条目是*(S,G)*形式的，*S*是源地址，*G*是多播组。*iif*是接收多播数据包的网络接口，如果设备名是关键词*unresolved*，就表示路由监控进程不能解析这个条目；接下来的关键词是*oif*，它后面跟着一些输出网络接口，接口之间用空格分开。

统计信息

使用*-statistics*选项，我们可以得到更为详细的输出信息，包括：数据包的数量，通过这条路由转发的字节数以及到达错误接口的数据包数量(如果有)。

```
kuznet@amber:~ $ ip -s mr ls 224.66/16

(193.233.7.65, 224.66.66.66)     Iif: eth0       Oifs: pimreg
  9383 packets, 300256 bytes
kuznet@amber:~ $
```

11.ip tunnel -- 通道配置

11.1.缩写

tunnel、tunl

11.2.对象

*ip tunnel*命令的操作对象是网络通道(tunnel)。所谓通道(tunnel)是指把数据包封装到IPv4数据包中，使用IP协议发出。有关通道的更多信息，请参考iproute的文档*Tunnels over IP in Linux-2.2*。

11.3.命令

add、delete、change、show或者list

11.4.ip tunnel add -- 添加新的通道
ip tunnel change -- 修改现有的通道
ip tunnel delete -- 删除一个通道

缩写：add、a；change、chg；delete、del、d
参数

name NAME(default)	选择通道设备名
mode MODE	设置通道模式。有效的模式包括：*ipip*、*sit*和*gre*。
remote ADDRESS	设置通道远端地址
local ADDRESS	设置进入通道数据包的固定本地地址，必须是在本机另外一个接口上的地址。
ttl N	设置进入通道数据包的TTL为N。N是一个1—255之间的数字。0是一个特殊的值，表示这个数据包的TTL值是继承(*inherit*)的。ttl参数的缺省值是：*inherit*。

tos T或者dsfield T	设置进入通道数据包的TOS域，缺省是*inherit*。
dev NAME	把通道绑定到设备NAME，以便进入通道的数据包只能通过NAME设备路由，并且当对端发生变化时，不能够在另外的设备解开封装。
nopmtudisc	在这个通道上禁止*路径最大传输单元发现(Path MTU Discovery)*。默认情况下，这个功能是打开的。注意：这个选项和固定的ttl是不兼容的，如果使用了固定的*ttl*参数，系统会打开路径最大传输单元发现(Path MTU Discovery)功能。
key k,ikey k,okey k	只适用于GRE通道，设置keyed GRE通道的key。K或者是一个数字或者是IP地址形式的数字序列。参数key在通道的双向使用，ikey和okey为输入和输出设置不同的key。
csum,icsum,ocsum	只用于GRE通道，计算进入通道数据包的校验和。*ocsum*表示只计算出去的数据包的校验和；*icsum*表示只计算进入的数据包的校验和；而*csum*等于*icsum ocsum*。
seq,iseq,oseq	只适用于GRE通道，顺序发送/接收数据包。*oseq*使向外的数据包顺序发送；*iseq*要求所有进入的数据包都是按照顺序的；而*seq*等于*iseq oseq*。

示例

建立一个点对点通道，最大TTL是32

```
netadm@amber:~ # ip tunnel add Cisco mode sit remote
192.31.7.104
                                                local
192.203.80.1 ttl 32
```

11.5.ip tunnel show -- 列出现有的通道

缩写：show、list、sh、ls、l

参数

无

输出格式

```
kuznet@amber:~ $ ip tunnel ls Cisco

Cisco: ipv6/ip  remote 192.31.7.104  local 192.203.80.142  ttl
32
```

```
kuznet@amber:~ $
```

输出的第一部分是通道的设备名，接着是通道模式。下面就是设置通道时的各个参数。

统计信息

```
kuznet@amber:~ $ ip -s tunl ls Cisco

Cisco: ipv6/ip  remote 192.31.7.104  local 192.203.80.142  ttl
32
RX: Packets    Bytes        Errors CsumErrs OutOfSeq Mcasts
    12566      1707516      0      0        0        0
TX: Packets    Bytes        Errors DeadLoop NoRoute  NoBufs
    13445      1879677      0      0        0        0
kuznet@amber:~ $
```

以上输出结果里面的数字和使用 *ip -s link show* 的输出是一样的，但是每个标志都是特定于通道的。

CsumErrs	对于打开校验和检验的 GRE 通道，这个数字是由于校验和错误而丢弃的数据包数量。
OutOfSeg	在打开顺序功能的 GRE 通道内，由于顺序错误而丢弃的数据包数量。
Mcasts	在 GRE 通道上接收到的多播数据包的数量。
DeadLoop	由于通道是回环到自己而没有传输的数据包数目。
NoRoute	由于到对端没有路由而没有被传输的数据包数目。
NoBufs	由于内核不能分配缓冲区而没有被传输的数据包数目。

12.ip monitor 和 rtmon -- 状态监视

ip 命令可以用于连续地监视设备、地址和路由的状态。这个命令选项的格式有点不同，命令选项的名字叫做 *monitor*，接着是操作对象：

```
ip monitor [ file FILE ] [ all | OBJECT-LIST ]
```

OBJECT-LIST 是一些被监控的对象，它可以包括 link、address 和 route。如果没有给出 *file* 参数，ip 命令就打开 RTNETLINK，在上面监听，并把状态的变化输出到标准输出设备。

如果使用了 *file* 参数，ip 命令就不是在 RTNETLINK 上监听，而是打开由 *file* 参数指定的包含 RTNETLINK 信息的二进制文件，把解析的结果显示出来。这

种历史文件可以有工具产生。这个工具具有和 *ip monitor* 命令的语法类似的命令行。理想的情况是，在网络配置命令起动之前运行 *rtmon* 命令(当然，你可以在任意的时间起动 *rtmon*，它会记录从起动开始的状态变化)。你可以在起动脚本中插入以下命令行：

```
rtmon file /var/log/rtmon.log
```

如果我们执行如下命令：

```
[root@nixeOn root]ip route add dev eth0 to 61.133.4.7 via
211.99.114.65

[root@nixeOn root]ip route del dev eth0 to 61.133.4.7
```

然后，我们使用 *ip monitor* 命令分析/var/log/rtmon.log 会得到如下输出结果：

```
[root@nixeOn root]ip monitor file /var/log/rtmon.log r

Timestamp: Wed Nov  6 20:25:54 2002 733331 us
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast
    link/ether 00:01:4f:00:15:f1 brd ff:ff:ff:ff:ff:ff
Timestamp: Wed Nov  6 20:25:58 2002 33700 us
61.133.4.7 via 211.99.114.65 dev eth0
Timestamp: Wed Nov  6 20:25:59 2002 924124 us
Deleted 61.133.4.7 via 211.99.114.65 dev eth0
[root@nixeOn root]
```

13.rtacct -- 路由范围和策略传播

在使用 OSPF 或者 BGP 协议的路由器上，其路由表可能会很大。如果我们需要对其进行归类或者计算通过每条路由的数据包，就需要保留很多信息。更糟糕的是，如果我们需要区别的不止是数据包的目的地址，还要包括它们的源地址，这个任务就更为复杂了，几乎无法解决。

对于这个问题，Cisco IOS Release 12.0 Quality of Service Solutions Configuration Guide: Configuring QoS Policy Propagation via Border Gateway Protocol 提出了一个解决方案，就是把策略从路由协议迁移到转发引擎。基本上，通过 BGP 的 Cisco 策略迁移(Cisco Policy Propagation via BGP)就是基于此种方式，它使路由器保留所有和转发引擎关系紧密的 RIB(Routing Information Base，路由信息库)，以便策略路由规则能够监查所有的路由属性，包括 ASPATH 的信息和团体(community)字符串。

而Linux把这分为由用户空间监控维护的路由信息库(Routing Infomation Base,RIB)，和内核层的转发信息库(Forwarding Infomation Base,FIB)。

这是我们的幸运，因为还有另外的解决方案，而这个方案允许更为灵活的策略和更为丰富的语义。

换句话说，可以在用户空间根据路由的属性把它们归类，例如：BGP路由的ASPATH、团体(community)；OSPF路由的标记和它们的范围。而管理员手工添加路由时，也知道它们的属性。按照这个标准划分的集合(我们把它们叫做*realm*)数量就很少了，因此按照路由的源地址和目的地址进行完全的分类就可以管理了。

因此，每个路由都可以被分配到一个*范围(realm)*中。一般这是有路由监控进程作的，不过对于静态路由，也可以使用*ip route*命令手工处理。

在某些情况下（例如路由监控进程不理解*realm*）为了方便，漏掉的*realm*可以由路由策略规则补齐。

内核使用如下算法计算每个数据包的源范围(*realm*)和目的范围(*realm*)：

```
    If route has a realm, destination realm of the packet is
set to it.

    If rule has a source realm, source realm of the packet is
set to it. If destination realm was not get from route and
rule has destination realm, it is also set.
    If at least one of realms is still unknown, kernel finds
reversed route to the source of the packet.
    If source realm is still unknown, get it from reversed
route.
    If one of realms is still unknown, swap realms of reversed
routes and apply step 2 again.
```

这个过程完成后，我们就知道了数据包的源范围和目的范围。如果某些还是未知，它就会被设置为0(realm unknown)

*范围(realm)*主要还是由TC(Traffic Control)的路由类别(*route classifier*)使用，我们可以使用路由类别把数据包分配到给不同的流量类(trafffic class)，为数据包计数，以及为它们制定调度策略。

相对于TC，使用realm为进入的数据包计数就简单多了，但这是一个非常有用的应用。内核可以根据*realm*收集总结数据包统计信息。在用户空间，我们可以使用工具*rtacct*查看这些信息。例如：

```
kuznet@amber:~ $ rtacct russia

Realm      BytesTo    PktsTo     BytesFrom  PktsFrom
russia     20576778   169176     47080168   153805
kuznet@amber:~ $
```

结果表示，这个路由器收到153805个来自*russia*地区的数据包，并且向*russia*转发了169176个数据包。*russia*范围由ASPATH(路径自治系统)在俄罗斯的路由组成。

14.参考


```
    T. Narten, E. Nordmark, W. Simpson. ``Neighbor Discovery for
IP Version 6 (IPv6)'', RFC-2461.


    S. Thomson, T. Narten. ``IPv6 Stateless Address
Autoconfiguration'', RFC-2462.
    F. Baker. ``Requirements for IP Version 4 Routers'', RFC-
1812.
    R. T. Braden. ``Requirements for Internet hosts --
communication layers'', RFC-1122.
    ``Cisco IOS Release 12.0 Network Protocols Command
Reference, Part 1'' and ``Cisco IOS Release 12.0 Quality of
Service Solutions Configuration Guide: Configuring Policy-Based
Routing'',http://www.cisco.com/univercd/cc/td/doc/product/softwa
re/ios120.
    A. N. Kuznetsov. ``Tunnels over IP in Linux-2.2'',在:
ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz.
    A. N. Kuznetsov. ``TC Command Reference'',在:
ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz.
    ``Cisco IOS Release 12.0 Quality of Service Solutions
Configuration Guide: Configuring QoS Policy Propagation via
Border Gateway
Protocol'',http://www.cisco.com/univercd/cc/td/doc/product/softw
are/ios120.
    R. Droms. ``Dynamic Host Configuration Protocol.'', RFC-2131
```