第 5 章

管理虚拟网络

本章将系统地讲解虚拟网络的日常管理，这需要具备一定的 Linux 网络管理的知识，包括物理网络接口、虚拟网络接口、网桥及路由等。

本章要点：

- TUN/TAP 设备工作原理与管理。
- 网桥工作原理与管理。
- 理解不同网络类型的原理。
- 掌握 NAT、桥接、隔离、路由、开放等网络类型的配置。
- 掌握 VLAN 的原理与配置。
- 掌握网络过滤器的原理与配置。

5.1 查看默认网络环境

KVM 支持多种网络类型，首先我们查看一下默认的网络环境。

在 RHEL/CentOS 8 等 Linux 发行版本中安装虚拟化组件的时候，通常会自动创建一个默认的虚拟网络配置，这包括一个名为 virbr0 的网桥、virbr0-nic 的虚拟网络接口、iptables 的 NAT 配置及 DNSMASQ 的配置等。下面我们就通过实验来查看默认的网络环境，从而理解 libvirt 虚拟网络的原理。

5.1.1 查看宿主机的网络环境

在没有启动虚拟机的情况下，宿主机上默认的网络环境如图 5-1 所示。

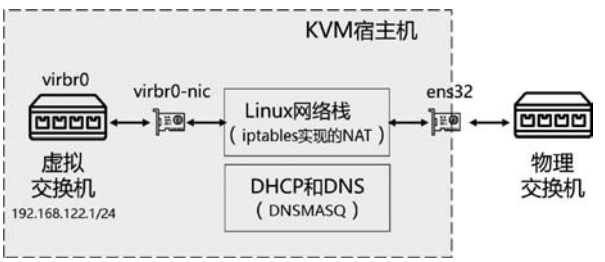

图 5-1 宿主机上默认的网络环境

查看宿主机网络环境的命令如下：

```
1 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
         valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
         valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:f7:6b:c8 brd ff:ff:ff:ff:ff:ff
      inet 192.168.114.231/24 brd 192.168.114.255 scope global noprefixroute ens32
         valid_lft forever preferred_lft forever
      inet6 fe80::20c:29ff:fef7:6bc8/64 scope link
         valid_lft forever preferred_lft forever
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
      inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
         valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff

2 # ip link
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
      link/ether 00:0c:29:f7:6b:c8 brd ff:ff:ff:ff:ff:ff
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode
DEFAULT group default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN mode
DEFAULT group default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff

3 # nmcli connection
  NAME     UUID                                   TYPE       DEVICE
  ens32    152beb06-47c5-c5e8-95a9-385590654382   ethernet   ens32
  virbr0   3068816f-b57c-4620-aea2-0e83642cdd87   bridge     virbr0

4 # nmcli device
  DEVICE          TYPE       STATE        CONNECTION
  ens32           ethernet   connected    ens32
  virbr0          bridge     connected    virbr0
  lo              loopback   unmanaged    --
  virbr0-nic      tun        unmanaged    --
```

libvirt 网络的最重要的组件是虚拟网络交换机，默认为由 Linux 的网桥实现。libvirt 默认会创建一个名为 virbr0 的网桥，virbr0 是 Virtual Bridge 0 的缩写。与物理交换机类似，虚拟网络交换机也是从其接收的数据包（帧）中获得 MAC 地址，并存储在 MAC 表中。物理交换机的端口数量有限，而虚拟交换机的端口数量则没有限制。

在 Linux 系统中，可以向网桥分配 IP 地址。从第 1 行命令的输出中可以看到，libvirt 向 virbr0 分配的 IP 地址是 192.168.122.1/24。libvirt 根据自己的配置文件来分配这个 IP 地址，所以不会在/etc/sysconfig/network-scripts/目录中看到名为 ifcfg- virbr0 的配置文件。

由于 Linux 的网桥会将其上第 1 个接口设备的 MAC 地址当作它的 MAC 地址，所以 libvirt 会创建一个名为 virbr0-nic 的 TAP 设备。从第 1 行、第 2 行命令的输出中可以看出 virbr0 与 virbr0-nic 的 MAC 地址都是 52:54:00:e0:41:ac。

连接网桥的接口通常被称为 slave 接口。它们既可以是物理网卡，也可以是虚拟网络接口设备（例如：TAP 类型的虚拟设备），所以通过网桥可以连通物理与虚拟网络设备。从相互关系上来讲，这些 slave 接口的 master 就是虚拟机交换机。从第 1 行、第 2 行命令的输出中可以看到 virbr0-nic 的属性字符串有 master virbr0 字样，这说明它是 virbr0 网桥的 slave 接口。

5.1.2　查看 libvirt 的网络环境

在 libvirtd 守护程序启动时，会根据配置文件创建一个名为 default 的虚拟网络。除了在宿主机上创建 virbr0、virbr0-nic 之外，还会通过配置 IP 转发、iptables 的 NAT 表从而在 Linux 协议栈中实现 NAT 功能。

libvirt 使用的是 IP 伪装（IP masquerading），而不是源 NAT（Source-NAT，SNAT）或目标 NAT（Destination-NAT，DNAT）。IP 伪装使虚拟机可以使用宿主机的 IP 地址与外部网络进行通信。默认情况下，当虚拟网络交换机以 NAT 模式运行时，虚拟机可以访问位于宿主物理计算机外部的资源，但是位于宿主物理计算机外部的计算机无法与内部的虚拟机进行通信，也就是说：仅允许由内到外的访问，而不允许从外到内的访问。

首先，我们通过 virt-manager 这种比较直观的方式来查看虚拟网络 default：

（1）在 virt-manager 的 Edit 菜单上，选择 Connection Details。

（2）在打开的 Connection Details 菜单中单击 Virtual Networks 选项卡，如图 5-2 所示。

（3）窗口的左侧列出了所有可用的虚拟网络。我们可以查看和修改虚拟网络的配置。

下面通过命令行工具查看网络环境，命令如下：

```
1 #cat /proc/sys/net/ipv4/ip_forward
  1

2 #iptables -t nat -L -n
  Chain PREROUTING (policy ACCEPT)
  target     prot opt source               destination
```

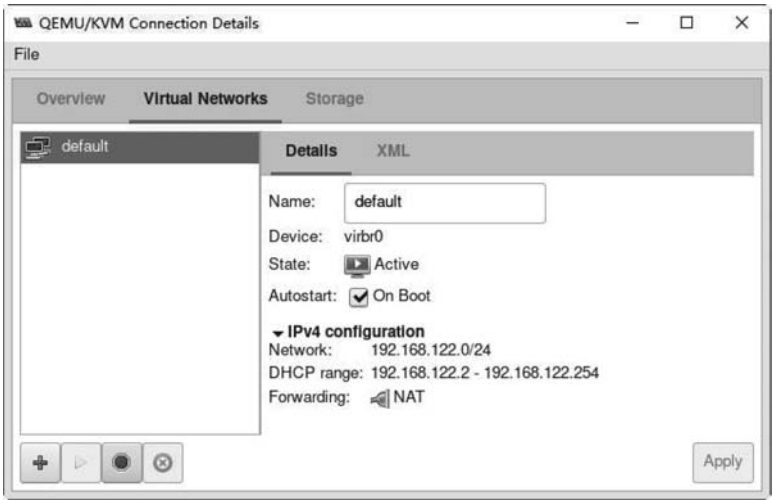

图 5-2　virt-manager 中的虚拟网络配置

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
RETURN     all  --  192.168.122.0/24     224.0.0.0/24
RETURN     all  --  192.168.122.0/24     255.255.255.255
MASQUERADE  tcp  --  192.168.122.0/24     !192.168.122.0/24   masq ports: 1024-65535
MASQUERADE  udp  --  192.168.122.0/24     !192.168.122.0/24   masq ports: 1024-65535
MASQUERADE  all  --  192.168.122.0/24     !192.168.122.0/24

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

注意：不建议在虚拟交换机运行时编辑这些防火墙规则，因为有可能导致交换机通信故障。

为了简化虚拟网络中的管理，libvirt 还使用了 DNSMASQ 组件为 default 网络中的虚拟机提供 DNS 和 DHCP 功能，命令如下：

```
3 # ps aux | grep dnsmasq
  dnsmasq      1561  0.0  0.0  71888  2428 ?        S    08:21    0:00
/usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf
--leasefile-ro --dhcp-script=/usr/libexec/libvirt_leaseshelper
  root         1562  0.0  0.0  71860  1828 ?        S    08:21    0:00
/usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf
```

```
-- leasefile-ro -- dhcp-script=/usr/libexec/libvirt_leaseshelper
  root       3041  0.0  0.0  12108  1108  pts/2     S+   09:24     0:00 grep
-- color=auto dnsmasq

4 #cat /var/lib/libvirt/dnsmasq/default.conf
  ##WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
  ##OVERWRITTEN AND LOST. Changes to this configuration should be made using:
  ## virsh net-edit default
  ##or other application using the libvirt API.
  ##
  ##dnsmasq conf file created by libvirt
  strict-order
  pid-file=/var/run/libvirt/network/default.pid
  except-interface=lo
  bind-dynamic
  interface=virbr0
  dhcp-range=192.168.122.2,192.168.122.254
  dhcp-no-override
  dhcp-authoritative
  dhcp-lease-max=253
  dhcp-hostsfile=/var/lib/libvirt/dnsmasq/default.hostsfile
  addn-hosts=/var/lib/libvirt/dnsmasq/default.addnhosts
```

从第3行命令的输出中可以看出DNSMASQ所使用的配置文件也是由libvirt提供的，它的路径名为/var/lib/libvirt/dnsmasq/default.conf。

从第4行命令的输出中可以看出向虚拟机分配的IP地址的范围是从192.168.122.2到192.168.122.254。

接下来，执行如下命令：

```
5 #virsh net-list
   Name          State     Autostart     Persistent
  ----------------------------------------------------------
   default       active    yes           yes

6 #virsh net-dumpxml default
  <network connections='2'>
    <name>default</name>
    <uuid>6c729bec-ce6c-4ca3-b05c-1fdb99be4fdc</uuid>
    <forward mode='nat'>
      <nat>
        <port start='1024' end='65535'/>
      </nat>
    </forward>
    <bridge name='virbr0' stp='on' delay='0'/>
```

```
    <mac address = '52:54:00:e0:41:ac'/>
    <ip address = '192.168.122.1' netmask = '255.255.255.0'>
      <dhcp>
        <range start = '192.168.122.2' end = '192.168.122.254'/>
      </dhcp>
    </ip>
  </network>

7 # ls /etc/libvirt/qemu/networks/
  autostart default.xml

8 # cat /etc/libvirt/qemu/networks/default.xml
  <!--
  WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
  OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
    virsh net-edit default
  or other application using the libvirt API.
  -->

  <network>
    <name>default</name>
    <uuid>6c729bec-ce6c-4ca3-b05c-1fdb99be4fdc</uuid>
    <forward mode = 'nat'/>
    <bridge name = 'virbr0' stp = 'on' delay = '0'/>
    <mac address = '52:54:00:e0:41:ac'/>
    <ip address = '192.168.122.1' netmask = '255.255.255.0'>
      <dhcp>
        <range start = '192.168.122.2' end = '192.168.122.254'/>
      </dhcp>
    </ip>
  </network>
```

从第 5 行命令的输出可以看出，这个名为 default 的网络是随着 libvirtd 的启动而自动启动的，当前的状态是已激活。

第 6 行命令输出了 default 网络的详细定义：

（1）<name>default</name>指定了虚拟网络的名称。

（2）<uuid>6c729bec-ce6c-4ca3-b05c-1fdb99be4fdc</uuid>指定了虚拟网络的全局唯一标识符。

（3）<forward mode='nat'>指定了虚拟网络将连接到物理网络，mode 属性确定了转发方法，目前允许的方法有 nat、route、open、bridge、private、vepa、passthrough 和 hostdev。如果没有配置 forward 属性，则该网络与任何其他网络都是隔离的，也就是 isolated 模式。nat 会在连接到该网络的虚拟机与物理网络之间进行网络地址转换。

（4）<port start='1024' end='65535'/>设置了用于<nat>的端口范围。

(5) <bridge name='virbr0' stp='on' delay='0'/>指定了 libvirt 在宿主机上创建网桥设备的信息。name 属性定义了网桥设备的名称，新网桥启用对生成树协议(STP)的支持，默认延迟为 0。建议使用以 virbr 开头的桥名称。虚拟机连接到该桥接设备，就像将真实世界的计算机连接到物理交换机一样。

(6) <mac address='52:54:00:e0:41:ac'/>属性定义了一个 MAC(硬件)地址，格式为 6 组 2 位十六进制数字，各组之间用冒号分隔。这个 MAC 地址在创建时即分配给桥接设备。建议让 libvirt 自动生成一个随机 MAC 地址并将其保存在配置中。

(7) <ip address='192.168.122.1' netmask='255.255.255.0'>为网桥指定 IP 地址。

(8) <dhcp>指定了在虚拟网络上启用 DHCP 服务。

(9) <range start='192.168.122.2' end='192.168.122.254'/>指定了要提供给 DHCP 客户端的地址池的边界。

提示：libvirt 网络配置的详细介绍可参见 https://libvirt.org/formatnetwork.html。

RHEL/CentOS 8 中的 libvirt 网络配置保存在/etc/libvirt/qemu/networks/目录下的 XML 文件中。强烈不建议直接编辑这些配置文件，而应通过 virsh 的 net-edit 子命令来修改。

5.1.3　查看虚拟机的网络配置

下面，我们再查看一下当启动虚拟机后网络所发生的变化，命令如下：

```
1 # virsh domiflist centos6.10
  Interface    Type       Source    Model    MAC
  -------------------------------------------------------
  -            network    default   virtio   52:54:00:27:5f:c9

2 # virsh start centos6.10

3 # virsh domiflist centos6.10
  Interface    Type       Source    Model    MAC
  -------------------------------------------------------
  vnet0        network    default   virtio   52:54:00:27:5f:c9
```

从第 1 行命令的输出中可以看出：这个名为 centos6.10 的虚拟机未启动之时，Interface 的属性为空，它连接到网络的名称是 default。

当启动此虚拟机时，libvirt 会在宿主机上创建一个新的虚拟网络接口，并将其连接到网桥中的 virbr0 上。我们可以从系统日志(/var/log/messages)中看到类似这样的信息：

```
Kernel: virbr0: port 2(vnet0) entered blocking state
Kernel: virbr0: port 2(vnet0) entered disabled state
Kernel: device vnet0 entered promiscuous mode
```

新的虚拟网络接口的名称是以 vnet 开头的，后面是从 0 开始的序号。如果虚拟机操作系统的网络是自动获得 IP 地址的，则还会在系统日志中看到它从 DNSMASQ 中租用 IP 地址的信息：

```
dnsmasq-dhcp[1561]: DHCPDISCOVER(virbr0) 192.168.122.142 52:54:00:27:5f:c9
dnsmasq-dhcp[1561]: DHCPOFFER(virbr0) 192.168.122.142 52:54:00:27:5f:c9
dnsmasq-dhcp[1561]: DHCPREQUEST(virbr0) 192.168.122.142 52:54:00:27:5f:c9
dnsmasq-dhcp[1561]: DHCPACK(virbr0) 192.168.122.142 52:54:00:27:5f:c9
```

我们还可以使用 ip 和 nmcli 命令查看宿主机上网络的变化情况，命令如下：

```
4 # ip addr
    ...
  3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
      inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
         valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  5: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN group default qlen 1000
      link/ether fe:54:00:27:5f:c9 brd ff:ff:ff:ff:ff:ff
      inet6 fe80::fc54:ff:fe27:5fc9/64 scope link
         valid_lft forever preferred_lft forever

5 # nmcli connection
  NAME    UUID                                  TYPE      DEVICE
  ens32   152beb06-47c5-c5e8-95a9-385590654382  ethernet  ens32
  virbr0  3068816f-b57c-4620-aea2-0e83642cdd87  bridge    virbr0
  vnet0   0c886394-b825-432f-9f79-bfe0a67199db  tun       vnet0
```

从第 4 行命令的输出可以看出：新增加了一个名 vnet0 的接口，它的 master 是 virbr0。第 5 行命令的输出显示了它的类型是 TUN，准确来讲应当是 TAP 设备。

注意：NetworkManager 有些版本（例如 1.22.8-5.el8_2.x86_64）总是将 TUN/TAP 设备显示为 TAP 设备，而 ip 命令（例如 iproute-5.3.0-1.el8.x86_64）显示的类型则是正确的。5.2 节将深入讲解 TUN/TAP 设备。

我们再启动一台新的虚拟机 win2k3，它会在网桥 virbr0 中再新增加一个 slave 设备，新设备名称为 vnet1，命令如下：

```
6 # ip addr
    …
  5: vnet0: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN group default qlen 1000
      link/ether fe:54:00:27:5f:c9 brd ff:ff:ff:ff:ff:ff
      inet6 fe80::fc54:ff:fe27:5fc9/64 scope link
         valid_lft forever preferred_lft forever
  6: vnet1: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN group default qlen 1000
      link/ether fe:54:00:9d:57:9c brd ff:ff:ff:ff:ff:ff
      inet6 fe80::fc54:ff:fe9d:579c/64 scope link
         valid_lft forever preferred_lft forever

7 # nmcli connection
  NAME     UUID                                   TYPE       DEVICE
  ens32    152beb06 - 47c5 - c5e8 - 95a9 - 385590654382   ethernet   ens32
  virbr0   3068816f - b57c - 4620 - aea2 - 0e83642cdd87   bridge     virbr0
  vnet0    0c886394 - b825 - 432f - 9f79 - bfe0a67199db   tun        vnet0
  vnet1    3d4e1388 - aee6 - 42df - a283 - fcc909939762   tun        vnet1
```

此时宿主机上网络环境如图 5-3 所示。两台虚拟机之间的通信、宿主机与虚拟机之间的通信全部是在虚拟交换机(网桥)virbr0 内部完成的。两台虚拟机对外部网络的访问是先将消息发送给它们的默认网关 virbr0(192.168.122.1),再通过 iptables 实现的 NAT 功能从 ens32 发送出去。由于采用的 NAT 模式是 MASQUERADE(一种 SNAT),所以默认情况下外部网络的主机不能访问这两台虚拟机,换句话来讲,就是这两台虚拟机对于外部网络来讲是不可见的。

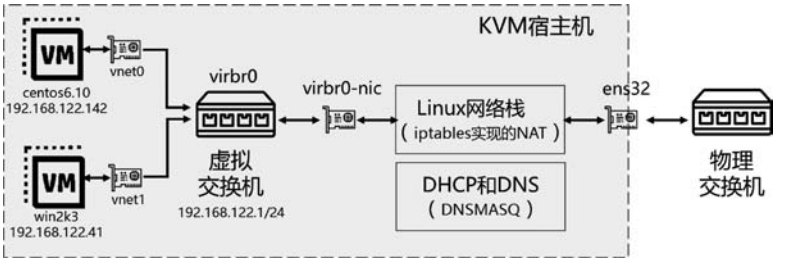

图 5-3　RHEL/CentOS8 中 libvirt 使用的 default 网络的工作原理

5.2　TUN/TAP 设备工作原理与管理

libvirt 的虚拟网络离不开 TAP 类型的虚拟网络接口设备,所以需要讲解一下它的工作原理。首先回顾一下 Linux 物理网卡的工作原理。计算机系统通常会有一个或多个网络设备,例如 eth0、eth1 等。这些网络设备与物理网络适配器相关联,后者负责将数据包放置到

线路上，如图 5-4 所示。

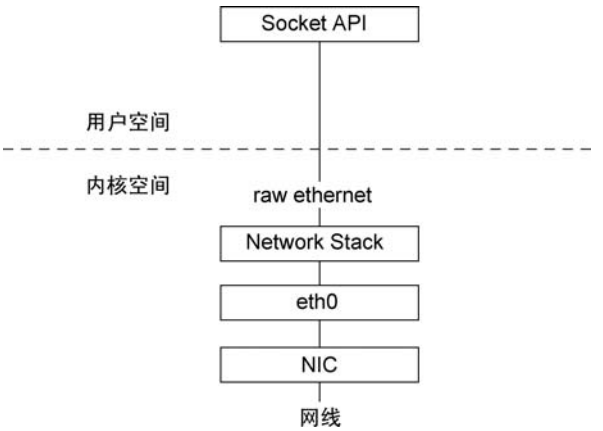

图 5-4　Linux 物理网卡工作原理

eth0 通过物理网卡 NIC 与外部网络相连，该物理网卡收到的数据包会经由 eth0 传递给内核的网络协议栈(Network Stack)，然后由协议栈对这些数据包进行进一步的处理。

对于一些错误的数据包，协议栈可以选择丢弃。对于目标不属于本机的数据包，协议栈可以选择转发，而对于目标地址是本机的而且是上层应用所需要数据包，协议栈会通过 Socket API 传递给上层正在等待的应用程序。

虚拟网络接口设备完成的功能与物理网卡类似，有两种常见类型：TUN 和 TAP。在 Linux 内核官方网站上有一个文档(https://www.Kernel.org/doc/Documentation/networking/tuntap.txt)是这样定义的：TUN/TAP 为用户空间程序提供数据包的接收和传输。可以将其视为简单的点对点或以太网设备。它们不是从物理介质接收数据包，而是从用户空间程序接收数据包。它们也不是通过物理介质发送数据包，而是将其写入用户空间程序。

也就是说，TUN/TAP 接口是没有关联物理设备的虚拟接口。用户空间程序可以附加到 TUN/TAP 接口并处理发送到该接口的流量，如图 5-5 所示。

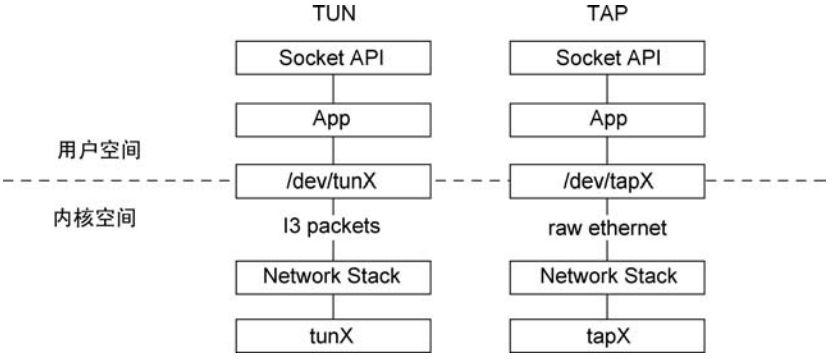

图 5-5　Linux 中 TUN/TAP 虚拟设备工作原理

使用 TUN/TAP 技术可以在主机上构建虚拟网络接口，它们的功能类似于物理网卡，可以为其分配 IP、将数据包路由到该接口、分析流量等。TUN/TAP 有两种常用的应用场景：VPN 和云计算。

TUN 与 TAP 有什么区别呢？

TUN(tunnel)设备在 OSI 模型的第 3 层运行，它实现的是虚拟的 IP 点对点接口，应用程序只能从此接口发送或接收 IP 数据包。它没有 MAC 地址，而且不能成为网桥的 slave 设备。

TAP(Network Tap)的运行方式与 TUN 极为相似，但是它运行在 OSI 模型的第 2 层上，应用程序从此接口发送和接收原始以太网数据包，所以它与以太网卡特别相似。它有 MAC 地址，可以成为网桥的 slave 设备。

下面通过实验讲解 TAP 设备的管理。

首先是 ip 命令。ip 是 iproute 软件包里面的一个强大的网络配置工具，其中 tuntap 子命令可用于管理 TUN/TAP 接口，命令如下：

```
1 # ip link
  1: lo: < LOOPBACK, UP, LOWER_UP > mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  2: ens32: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
    link/ether 00:0c:29:f7:6b:c8 brd ff:ff:ff:ff:ff:ff
  3: virbr0: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  4: virbr0 - nic: < BROADCAST, MULTICAST > mtu 1500 qdisc fq_codel master virbr0 state DOWN mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  6: vnet0: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:27:5f:c9 brd ff:ff:ff:ff:ff:ff
  7: vnet1: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:9d:57:9c brd ff:ff:ff:ff:ff:ff

2 # ip tuntap help
  Usage: ip tuntap { add | del | show | list | lst | help } [ dev PHYS_DEV ]
      [ mode { tun | tap } ] [ user USER ] [ group GROUP ]
      [ one_queue ] [ pi ] [ vnet_hdr ] [ multi_queue ] [ name NAME ]

  Where:USER : = { STRING | NUMBER }
      GROUP : = { STRING | NUMBER }
```

```
3 # ip tuntap list
  virbr0 - nic: tap persist
  vnet0: tap vnet_hdr
  vnet1: tap
```

第 1 行 ip link 命令用于显示当前主机上的所有物理及虚拟的网络设备。

第 2 行命令用于显示 ip 命令的 tuntap 子命令的帮助。

第 3 行命令用于显示当前的 TUN/TAP 设备信息，此处共有 3 个 TAP 设备。

提示：libvirt 在创建 TAP 设备时，会根据虚拟机网卡类型及用途为 TAP 设备增加额外的标记(Flag)，例如为 virtio 类型网卡增加的标记是 vnet_hdr，而对于传统的 e1000 的网卡则不设置标记。

接下来添加 1 个 TAP 设备，然后查看这个新设备，命令如下：

```
4 # ip tuntap add dev tap - nic1 mode tap

5 # ip tuntap list
  virbr0 - nic: tap persist
  vnet0: tap vnet_hdr
  vnet1: tap
  tap - nic1: tap persist

6 # ip link
  1: lo: < LOOPBACK, UP, LOWER_UP > mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  2: ens32: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
    link/ether 00:0c:29:f7:6b:c8 brd ff:ff:ff:ff:ff:ff
  3: virbr0: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  4: virbr0 - nic: < BROADCAST, MULTICAST > mtu 1500 qdisc fq_codel master virbr0 state DOWN mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  6: vnet0: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:27:5f:c9 brd ff:ff:ff:ff:ff:ff
  7: vnet1: < BROADCAST, MULTICAST, UP, LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:9d:57:9c brd ff:ff:ff:ff:ff:ff
  8: tap - nic1: < BROADCAST, MULTICAST > mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
```

```
    link/ether 76:0a:b9:fa:83:e9 brd ff:ff:ff:ff:ff:ff

7 # ip addr add 172.16.123.1/24 dev tap-nic1
```

第4行命令添加了1个名为tap-nic1的TAP设备。从第5、6行命令的输出可以看到这个新的设备。

我们可以像管理物理网卡一样管理虚拟网络接口，例如第7行命令给它设置了1个静态的IP地址。接下来删除这个TAP设备，然后查看是否成功删除，命令如下：

```
8 # ip tuntap del dev tap-nic1 mode tap

9 # ip tuntap list
  virbr0-nic: tap persist
  vnet0: tap vnet_hdr
  vnet1: tap
```

实验做完了，第8行命令用于删除这个TAP设备，第9行命令用于查看TAP设备列表。

RHEL/CentOS 8中使用ip命令配置设备信息，大部分会在设备重启后还原。如果需要持久的TUN/TAP设置，则推荐使用NetworkManager来创建。下面我们通过NetworkManager中的命令行工具nmcli来做类似的实验，命令如下：

```
10 # nmcli connection add type tun con-name newcon1 ifname tap-nic2 mode tap ip4 172.16.123.
2/24
   Connection 'newcon1' (09df2d02-dfc1-4a92-9d24-72dd97fe0756) successfully added.

11 # nmcli connection
   NAME     UUID                                  TYPE       DEVICE
   ens32    152beb06-47c5-c5e8-95a9-385590654382  ethernet   ens32
   newcon1  09df2d02-dfc1-4a92-9d24-72dd97fe0756  tun        tap-nic2
   virbr0   46edfc4d-8edb-4560-8a86-c84cad9b61c9  bridge     virbr0
   vnet0    540308fa-9ffa-40ec-8860-dd5ce13dde73  tun        vnet0
   vnet1    7a8486a7-88d0-4423-950a-0705d7aa3025  tun        vnet1

12 # nmcli device
   DEVICE       TYPE       STATE       CONNECTION
   ens32        ethernet   connected   ens32
   tap-nic2     tun        connected   newcon1
   virbr0       bridge     connected   virbr0
   vnet0        tun        connected   vnet0
   vnet1        tun        connected   vnet1
   lo           loopback   unmanaged   --
   virbr0-nic   tun        unmanaged   --
```

NetworkManager将所有网络配置存储为连接(connection),它们是描述如何创建或描述如何连接到网络的配置(2层详细信息、IP地址等)的集合。NetworkManager支持的连接类型有多种,包括ethernet、wifi、pppoe、infiniband、bluetooth、bond、bridge、tun、vxlan等。需要注意的是,NetworkManager不区分连接类型究竟是TUN还是TAP,统一称为TUN。

第10行命令创建了一个新的连接,将type设置为tun,通过con-name指定了连接的名称为newcon1,通过ifname指定了接口名称为tap-nic2,通过mode指定了模式为tap,通过ip4指定了IP地址为172.16.123.2/24。

从第11行命令的输出中可以看到这个名为newcon1的新连接及其属性。

从第12行命令的输出中可以看到新设备tap-nic2,type显示为tun。我们可以认为这是指连接的类型,而且不是设备的类型。tap-nic2的类型还是tap,这个可以从第14行命令的输出中看出,命令如下:

```
13 # ip address show tap-nic2
   7: tap-nic2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN
group default qlen 1000
       link/ether ce:ba:06:00:66:ef brd ff:ff:ff:ff:ff:ff
       inet 172.16.123.2/24 brd 172.16.123.255 scope global noprefixroute tap-nic2
          valid_lft forever preferred_lft forever
       inet6 fe80::c5d3:c6b8:d79:81a7/64 scope link tentative noprefixroute
          valid_lft forever preferred_lft forever

14 # ip tuntap
   virbr0-nic: tap persist
   vnet0: tap vnet_hdr
   vnet1: tap
   tap-nic2: tap persist

15 # nmcli connection delete newcon1
   Connection 'newcon1' (09df2d02-dfc1-4a92-9d24-72dd97fe0756) successfully deleted.

16 # ip tuntap list
   virbr0-nic: tap persist
   vnet0: tap vnet_hdr
   vnet1: tap
```

从第13行命令的输出中可看到虚拟网络接口tap-nic2的信息,包括随机生成的MAC地址、IPv4地址、IPv6地址等信息。

第15行命令会清除连接newcon1,这个操作也会同时删除虚拟网络接口设备tap-nic2。

5.3 网桥工作原理与管理

网桥是一种数据链路层的设备。2.4 版以后的 Linux 内核之中已经集成了网桥功能。在虚拟化解决方案中,网桥是一个很重要的组件。

5.3.1 考察现有网桥

在 RHEL/CentOS 8 中,可以通过 iproute、NetworkManager 和 Cockpit 来查看及了解当前主机上的网桥情况。

在“5.1.1 查看宿主机的网络环境”一节中讲解了如何通过 iproute 和 NetworkManager 来查看宿主机当前的网桥,下面我们通过 Cockpit 查看这部分配置。

在 Cockpit 中单击 Networking 就可以看到宿主机的所有网络接口信息,在其中可以看到网桥 virbr0,如图 5-6 所示。

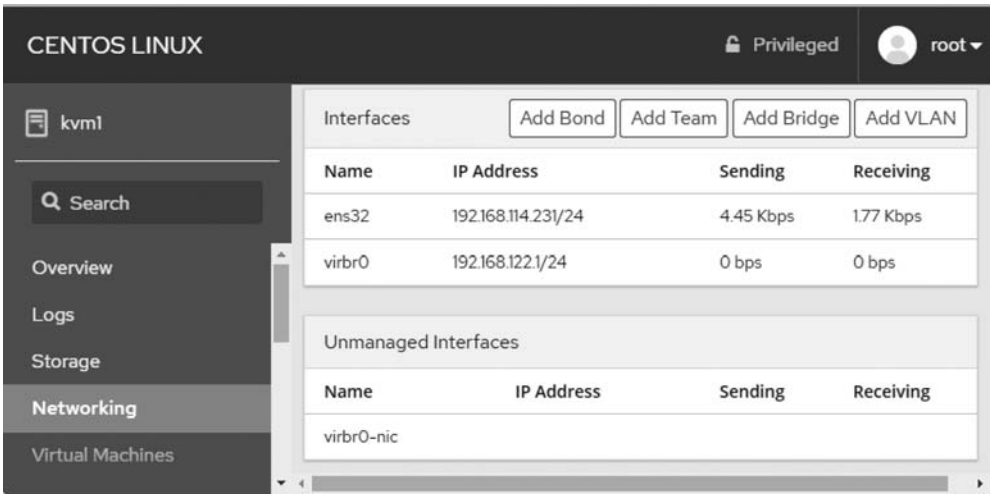

图 5-6 Cockpit 中的网络接口

单击 virbr0 就可以看到这个网桥的详细信息,包括 IP 地址、支持 STP、转发延迟及两个端口 vnet0 和 vnet1,如图 5-7 所示。

提示:RHEL 7.7 已经不推荐使用 bridge-utils 了,所以在 RHEL 8/CentOS 8 中已经无法使用 brctl 命令了,我们完全可以使用 iproute、NetworkManager 和 Cockpit 来替代它。

5.3.2 通过 iproute 管理网桥

在 iproute 软件包中,ip 命令是最常用的命令,下面通过它创建一个网桥接口,网桥接

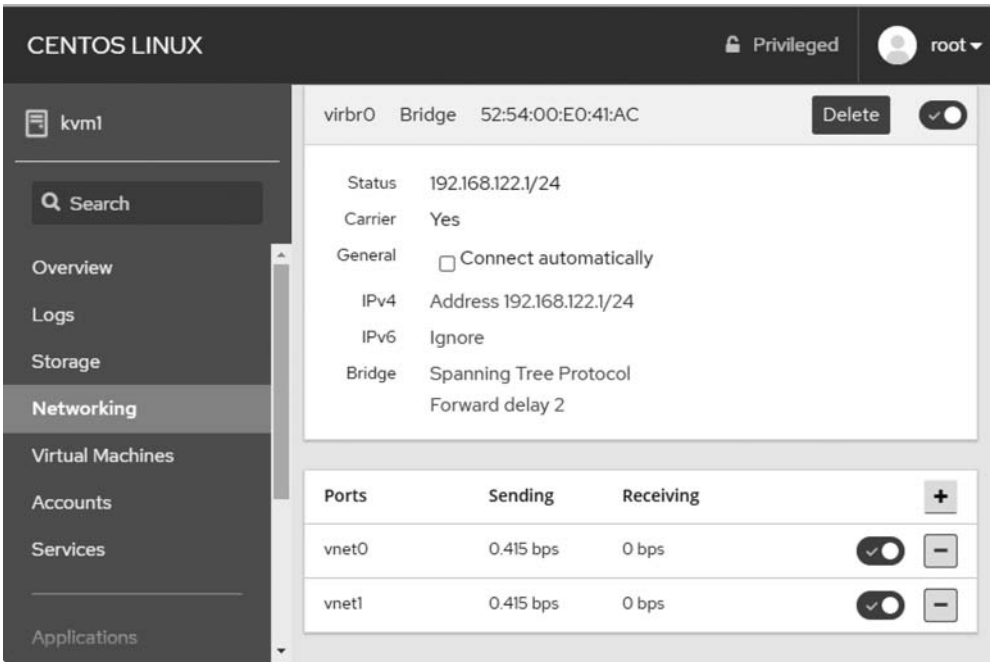

图 5-7 Cockpit 中的网桥详细信息

口的名称可以用于表示网桥，命令如下：

```
1 # ip link add name br2 type bridge

2 # ip address
  …
  7: br2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
      link/ether 36:21:b7:e4:58:fe brd ff:ff:ff:ff:ff:ff

3 # ip link
  …
  7: br2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen
1000
      link/ether 36:21:b7:e4:58:fe brd ff:ff:ff:ff:ff:ff
```

第 1 行命令创建了 1 个名为 br2 的网桥接口。从第 2 行、第 3 行命令的输出中可以看到这个新的网桥。网桥都有 1 个随机生成的 MAC 地址，注意此时这个新网桥的 MAC 地址是 36:21:b7:e4:58:fe。

有了网桥，下面就可以为其添加子接口了。网桥上的子接口既可以是物理接口也可以是虚拟接口，本次实验将使用虚拟接口，命令如下：

```
4 # ip tuntap add dev tap-nic1 mode tap

5 # ip tuntap add dev tun-nic1 mode tun

6 # ip tuntap
  virbr0-nic: tap persist
  vnet0: tap vnet_hdr
  vnet1: tap
  tap-nic1: tap persist
  tun-nic1: tun persist

7 # ip link
  …
  7: br2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen
1000
      link/ether 36:21:b7:e4:58:fe brd ff:ff:ff:ff:ff:ff
  8: tap-nic1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
      link/ether 26:3a:41:e8:7f:78 brd ff:ff:ff:ff:ff:ff
  9: tun-nic1: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 500
      link/none
```

第 4 行命令添加了 1 个名为 tap-nic1 的 TAP 类型的虚拟接口。出于对比的目的,使用第 5 行命令再创建了 1 个名为 tun-nic1 的 TUN 类型的虚拟接口。可以在第 6 行命令的输出中看到这 2 个虚拟接口。

从第 7 行命令的输出中可以看出,tap-nic1 与 br2 的 MAC 地址是不同的。由于 tun-nic1 是一个 3 层的设备,所以它是没有 MAC 地址的。

接下来激活连接及设置子接口,命令如下:

```
8 # ip link set br2 up

9 # ip link set tap-nic1 up

10 # ip link set tun-nic1 up

11 # ip link set tap-nic1 master br2

12 # ip link set tun-nic1 master br2
   RTNETLINK answers: Invalid argument
```

ip 命令创建的连接默认为没有被激活,所以第 8、9、10 行命令激活了这 3 个连接。第 11 行命令将这个名为 tap-nic1 的 TAP 设备设置为网桥 br2 的子接口,命令没有输出表示成功。由于 tun-nic1 是一个 3 层的设备,它无法充当网桥的子接口,所以第 12 行命令会返

回一个错误。

查看网络配置，命令如下：

```
13 #ip addr
  …
   3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
       link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
       inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
          valid_lft forever preferred_lft forever
   4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
       link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  …
   7: br2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
       link/ether 26:3a:41:e8:7f:78 brd ff:ff:ff:ff:ff:ff
       inet6 fe80::3421:b7ff:fee4:58fe/64 scope link
          valid_lft forever preferred_lft forever
   8: tap-nic1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel master br2
state DOWN group default qlen 1000
       link/ether 26:3a:41:e8:7f:78 brd ff:ff:ff:ff:ff:ff
   9: tun-nic1: <NO-CARRIER,POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc fq_codel state
DOWN group default qlen 500
       link/none
```

我们注意一下第 13 命令的输出：tap-nic1 的 master 是网桥 br2，而且网桥 br2 会将它的第 1 个接口的 MAC 地址当作自己的 MAC，也就是 tap-nic1 的 MAC 地址 26:3a:41:e8:7f:78。这是 Linux 网桥一个特性：它会自动将第 1 个网络接口的 MAC 当作自己的 MAC 地址，所以 libvirt 在创建网桥 virbr0 时，会先创建一个 TAP 类型的接口 virbr0-nic，这样 virbr0 和 virbr0-nic 的 MAC 地址都是 52:54:00:e0:41:ac。

提示：默认情况下 libvirt 生成的 MAC 地址是 52:54:00 开头的。

在 iproute 软件包中，除了 ip 命令外，还有 1 个名为 bridge 的命令也与网桥有关，命令如下：

```
14 #bridge link
   4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32
cost 100
   5: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master virbr0 state forwarding
priority 32 cost 100
   6: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master virbr0 state forwarding
priority 32 cost 100
```

```
    8: tap-nic1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 master br2 state disabled
priority 32 cost 100
```

做完实验之后,我们可以使用下述命令清理实验环境。

```
15 # ip link set tap-nic1 nomaster

16 # ip link delete br2 type bridge

17 # ip tuntap delete tap-nic1 mode tap

18 # ip tuntap delete tun-nic1 mode tun
```

5.3.3 通过 NetworkManager 管理网桥

可以通过 nmcli 的 connection 子命令来管理网桥。与 ip 命令类似,它也是先创建网桥的连接再创建子接口。不同之处是它会自动创建网桥接口文件。

在下面的实验中,我们将使用一块新网卡 ens34 作为新网桥的子接口,代码如下:

```
1 # nmcli connection
  NAME    UUID                                   TYPE       DEVICE
  ens32   152beb06-47c5-c5e8-95a9-385590654382   ethernet   ens32
  ens34   293e8ad9-d491-46ee-a0ac-789fc12c5407   ethernet   ens34
  virbr0  b4cc48df-f27f-4c97-9709-bcca97a890e5   bridge     virbr0
  vnet0   39e2fcf8-26f4-4e4f-be22-1eb79f43c01b   tun        vnet0
  vnet1   84718c08-1d46-4d0b-b80c-79fb448eccce   tun        vnet1

2 # nmcli device
  DEVICE         TYPE       STATE         CONNECTION
  ens32          ethernet   connected     ens32
  ens34          ethernet   connected     ens34
  virbr0         bridge     connected     virbr0
  vnet0          tun        connected     vnet0
  vnet1          tun        connected     vnet1
  lo             loopback   unmanaged     --
  virbr0-nic     tun        unmanaged     --

4 # ls /etc/sysconfig/network-scripts/
  ifcfg-ens32 ifcfg-ens34

5 # cat /etc/sysconfig/network-scripts/ifcfg-ens34
  TYPE=Ethernet
  PROXY_METHOD=none
  BROWSER_ONLY=no
```

```
  BOOTPROTO = dhcp
  DEFROUTE = yes
  IPV4_FAILURE_FATAL = no
  IPV6INIT = yes
  IPV6_AUTOCONF = yes
  IPV6_DEFROUTE = yes
  IPV6_FAILURE_FATAL = no
  IPV6_ADDR_GEN_MODE = stable - privacy
  NAME = ens34
  UUID = 293e8ad9 - d491 - 46ee - a0ac - 789fc12c5407
  DEVICE = ens34
  ONBOOT = yes

6 # ip address
  …
  3: ens34: < BROADCAST,MULTICAST,UP,LOWER_UP > mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
      inet 192.168.114.130/24 brd 192.168.114.255 scope global dynamic noprefixroute ens34
         valid_lft 1750sec preferred_lft 1750sec
      inet6 fe80::697b:329c:b7b3:238d/64 scope link noprefixroute
         valid_lft forever preferred_lft forever
  …
```

从第 1 行及第 2 行命令的输出中可以看出：有一个名为 ens34 的连接，它有一个同名的以太网接口设备。

在/etc/sysconfig/network-scripts/目录中有一个名为 ifcfg-ens34 的网桥接口文件，从第 5 行的命令输出就可以看到这个配置文件的内容。系统初始化时会自动根据这个配置文件进行相应的设置，例如通过 DHCP 给 ens34 分配的 IP 地址，这可以从第 6 行命令的输出中看到 IP 地址是 192.168.114.130/24。

提示：如果/etc/sysconfig/network-scripts/目录中没有新网卡的接口文件，则除了手工创建之外，还可以通过 nmcli device connect ens34 命令来生成一个新的接口文件。

接下来执行以下命令：

```
7 # nmcli connection add type bridge autoconnect yes con - name virbr1 ifname virbr1
   Connection 'virbr1' (24fc8f00 - ca86 - 47da - 817c - cb392c4d15b4) successfully added.

8 # nmcli connection
   NAME    UUID                                   TYPE       DEVICE
   virbr1  24fc8f00 - ca86 - 47da - 817c - cb392c4d15b4   bridge     virbr1
   ens32   152beb06 - 47c5 - c5e8 - 95a9 - 385590654382   ethernet   ens32
```

```
    ens34      861f3757-4b5b-464a-9d56-fd7c0b4e8cee      ethernet      ens34
    virbr0     d767930b-ee30-49ee-8d59-e51708f1cc1d      bridge        virbr0

9 # nmcli device
    DEVICE        TYPE        STATE                                  CONNECTION
    ens32         ethernet    connected                              ens32
    ens34         ethernet    connected                              ens34
    virbr0        bridge      connected                              virbr0
    virbr1        bridge      connecting (getting IP configuration)  virbr1
    lo            loopback    unmanaged                              --
    virbr0-nic    tun         unmanaged                              --

10 # ls /etc/sysconfig/network-scripts/
    ifcfg-ens32 ifcfg-ens34 ifcfg-virbr1

11 # cat /etc/sysconfig/network-scripts/ifcfg-virbr1
    STP=yes
    BRIDGING_OPTS=priority=32768
    TYPE=Bridge
    PROXY_METHOD=none
    BROWSER_ONLY=no
    BOOTPROTO=dhcp
    DEFROUTE=yes
    IPV4_FAILURE_FATAL=no
    IPV6INIT=yes
    IPV6_AUTOCONF=yes
    IPV6_DEFROUTE=yes
    IPV6_FAILURE_FATAL=no
    IPV6_ADDR_GEN_MODE=stable-privacy
    NAME=virbr1
    UUID=24fc8f00-ca86-47da-817c-cb392c4d15b4
    DEVICE=virbr1
    ONBOOT=yes
```

第7行命令成功添加了一个新的连接并设置了属性，type指定的类型是bridge，autoconnect指定为自动启动，con-name指定的连接名称为virbr1，ifname指定的接口名称也是virbr1。

从第8行、第9行的输出中可以看到这个网桥的连接及设备信息。

第7行命令会在/etc/sysconfig/network-scripts/目录中生成一个名为ifcfg-virbr1的网络接口文件。

注意第11行命令的输出中的两个属性：TYPE=Bridge和BOOTPROTO=dhcp。

接下来执行的命令如下：

```
12 # ip address
   …
    3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
        link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
        inet 192.168.114.130/24 brd 192.168.114.255 scope global dynamic noprefixroute ens34
           valid_lft 1582sec preferred_lft 1582sec
        inet6 fe80::697b:329c:b7b3:238d/64 scope link noprefixroute
           valid_lft forever preferred_lft forever
   …
    6: virbr1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
        link/ether 92:23:18:cb:00:e3 brd ff:ff:ff:ff:ff:ff

13 # nmcli connection modify virbr1 ipv4.addresses 172.16.123.11/24 ipv4.method manual

14 # cat /etc/sysconfig/network-scripts/ifcfg-virbr1
    STP=yes
    BRIDGING_OPTS=priority=32768
    TYPE=Bridge
    PROXY_METHOD=none
    BROWSER_ONLY=no
    BOOTPROTO=none
    DEFROUTE=yes
    IPV4_FAILURE_FATAL=no
    IPV6INIT=yes
    IPV6_AUTOCONF=yes
    IPV6_DEFROUTE=yes
    IPV6_FAILURE_FATAL=no
    IPV6_ADDR_GEN_MODE=stable-privacy
    NAME=virbr1
    UUID=24fc8f00-ca86-47da-817c-cb392c4d15b4
    DEVICE=virbr1
    ONBOOT=yes
    IPADDR=172.16.123.11
    PREFIX=24
```

从第 12 行命令的输出可以看出：由于还没有激活新网桥 virbr1，所以它没有从 DHCP 服务器获得 IP 地址。需要注意的是 virbr1 的 MAC 地址是随机生成的 92:23:18:cb:00:e3，与 ens34 的 MAC 地址 00:0c:29:f7:6b:d2 并不相同。

第 13 行命令将 virbr1 的 IP 地址修改为静态 IP 地址 172.16.123.11/24。这个修改会反映在/etc/sysconfig/network-scripts/目录中的网格接口文件 ifcfg-virbr1 的变化上。

接下来执行的命令如下：

```
15 # nmcli connection delete ens34
   Connection 'ens34'(861f3757-4b5b-464a-9d56-fd7c0b4e8cee) successfully deleted.

16 # nmcli connection add type bridge-slave autoconnect yes con-name ens34 master virbr1
   Connection 'ens34'(ad4ae2e5-f95f-415a-a9e1-23e2ff650956) successfully added.

17 # nmcli connection
   NAME     UUID                                  TYPE       DEVICE
   virbr1   24fc8f00-ca86-47da-817c-cb392c4d15b4  bridge     virbr1
   ens32    152beb06-47c5-c5e8-95a9-385590654382  ethernet   ens32
   virbr0   d767930b-ee30-49ee-8d59-e51708f1cc1d  bridge     virbr0
   ens34    ad4ae2e5-f95f-415a-a9e1-23e2ff650956  ethernet   ens34

18 # nmcli device
   DEVICE          TYPE       STATE        CONNECTION
   ens32           ethernet   connected    ens32
   virbr1          bridge     connected    virbr1
   virbr0          bridge     connected    virbr0
   ens34           ethernet   connected    ens34
   lo              loopback   unmanaged    --
   virbr0-nic      tun        unmanaged    --

19 # ls /etc/sysconfig/network-scripts/
   ifcfg-ens32 ifcfg-ens34 ifcfg-virbr1

20 # cat /etc/sysconfig/network-scripts/ifcfg-ens34
   TYPE=Ethernet
   NAME=ens34
   UUID=ad4ae2e5-f95f-415a-a9e1-23e2ff650956
   ONBOOT=yes
   BRIDGE=virbr1

21 # nmcli connection up virbr1
   Connection successfully activated (master waiting for slaves) (D-Bus active path: /org/
freedesktop/NetworkManager/ActiveConnection/7)
```

网络设备 ens34 现在还属于一个同名的连接，所以需要先将这个连接删除。第 15 行命令将连接 ens34 删除。

第 16 行命令将网络设备 ens34 添加到网桥 virbr1，从而成为网桥的子接口。type 指定的类型为 bridge-slave，autoconnect 指定为自动启动，con-name 指定的连接名称为 ens34，master 指定的网桥名称是 virbr1。

从第 17 行、第 18 行的输出中可以看到 virbr1、ens34 的连接及设备信息。

第16行命令会为ens34在/etc/sysconfig/network-scripts/目录中生成一个新的网络接口文件ifcfg-ens34。通过第20行命令查看这个新文件，新文件很简单，核心属性是BRIDGE=virbr1，它为ens34指定了master。

第21行命令激活了网桥。在RHEL/CentOS 8中，这些操作会在系统的/var/log/messages生成日志条目。当操作失败时，阅读这些日志条件将有助于我们进行排错操作。

接下来执行的命令如下：

```
22 # ip address
  …
    3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master virbr1 state
UP group default qlen 1000
        link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
  …
    6: virbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
        link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
        inet 172.16.123.11/24 brd 172.16.123.255 scope global noprefixroute virbr1
           valid_lft forever preferred_lft forever
        inet6 fe80::e708:9d2e:c0ca:f913/64 scope link noprefixroute
           valid_lft forever preferred_lft forever

23 # bridge link
    3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master virbr1 state forwarding
priority 32 cost 100
    5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32
cost 100
```

从第22行命令的输出可以看出：ens34的master是virbr1，这说明它是网桥virbr1的子接口。同时要注意virbr1的MAC地址已经不再是原有那个随机成的MAC地址了，而是借用了第1个子接口，即ens34的MAC地址00:0c:29:f7:6b:d2。

第23行命令的输出也显示出ens34的master是virbr1。

创建网桥的实验做完了，下面我们进行删除及恢复操作，命令如下：

```
24 # mkdir ~/bak

25 # cp /etc/sysconfig/network-scripts/ifcfg-* ~/bak

26 # nmcli connection down virbr1
   Connection 'virbr1' successfully deactivated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/7)

27 # nmcli connection delete virbr1
   Connection 'virbr1'(24fc8f00-ca86-47da-817c-cb392c4d15b4) successfully deleted.
```

```
28 #nmcli connection delete ens34
   Connection 'ens34' (ad4ae2e5 - f95f - 415a - a9e1 - 23e2ff650956) successfully deleted.

29 #nmcli device connect ens34
  Device 'ens34' successfully activated with '6d322e2c - af9c - 4c87 - 9740 - d4cf6e076bfc'.
```

为了后续的实验，使用第 24 行、第 25 行命令将当前的网桥接口文件备份到～/bak 目录中。

使用第 26 行命令关闭 virbr1，然后删除连接 virbr1、ens34。

通过第 29 行命令会重新连接设备 ens34，并在/etc/sysconfig/network-scripts/目录生成包含默认属性值的 ifcfg-ens34 文件。

提示：在 NetworkManager 中还有图形化网络连接编辑器 nm-connection-editor。与 nmclic 类似，它也可以很方便地添加、删除和修改 NetworkManager 存储的网络连接。

5.3.4　通过网络接口文件管理网桥

在目录/etc/sysconfig/network-scripts/下创建网桥的配置文件 ifcfg-virbr1 和子接口配置文件 ifcfg-ens34，也可以很方便地创建网桥，命令如下：

```
1 #vi /etc/sysconfig/network - scripts/ifcfg - virbr1
  #添加以下内容:
  TYPE = Bridge
  STP = yes
  BOOTPROTO = none
  NAME = virbr1
  DEVICE = virbr1
  ONBOOT = yes
  IPADDR = 172.16.123.11
  PREFIX = 24

2 #vi /etc/sysconfig/network - scripts/ifcfg - ens34
  #添加以下内容:
  TYPE = Ethernet
  NAME = ens34
  ONBOOT = yes
  BRIDGE = virbr1

3 #nmcli connection reload

4 #ip address
  …
```

```
  3: ens34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master virbr1 state UP
group default qlen 1000
      link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
  …
  6: virbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
      link/ether 00:0c:29:f7:6b:d2 brd ff:ff:ff:ff:ff:ff
      inet 172.16.123.11/24 brd 172.16.123.255 scope global noprefixroute virbr1
         valid_lft forever preferred_lft forever
      inet6 fe80::20c:29ff:feb3:7340/64 scope link
         valid_lft forever preferred_lft forever
```

如第 1 行、第 2 行命令所示，不管是网桥的配置文件还是子接口的配置文件，仅需要必需的属性即可。当然，还可以将上一实验备份在目录～/bak/下的 ifcfg-virbr1 和 ifcfg-ens34 直接复制到目录/etc/sysconfig/network-scripts/下。

当执行第 3 行命令重新加载网络连接配置之后，网桥就配置成功了。这可以从第 4 行命令的输出信息中得到验证。如果失败，则可以查看系统日志/var/log/messages 中的条目来查找错误原因。

```
5 # rm /etc/sysconfig/network - scripts/ifcfg - virbr1

6 # rm /etc/sysconfig/network - scripts/ifcfg - ens34

7 # nmcli connection down virbr1
  Connection ' virbr1 ' successfully deactivated ( D - Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/6)

8 # nmcli connection reload
```

清除网桥 virbr1 也很简单，首先执行第 5 行、第 6 行命令删除网桥接口配置文件，然后在关闭网桥 virbr1 之后重新加载网络连接即可。

5.3.5 通过 Cockpit 管理网桥

相对于前面几种方法，使用 Cockpit 来管理网桥是最简单的。在网络信息中单击 Add Bridge 就可以添加新网桥了，如图 5-8 所示。

输入新网桥名称 virbr1，选中子接口（Cockpit 将其称为 port）ens34，其他选项保持默认，然后单击 Apply 按钮创建新网桥，如图 5-9 所示。

通过 Cockpit 来创建新网桥 virbr1，会自动在/etc/sysconfig/network-scripts/目录中创建配置文件 ifcfg-ens34 和 ifcfg-virbr1。

Cockpit 默认使用 DHCP 服务向新网桥分配 IP 地址，当然也可以单击新网桥 virbr1 的链接进行修改，如图 5-10 所示。

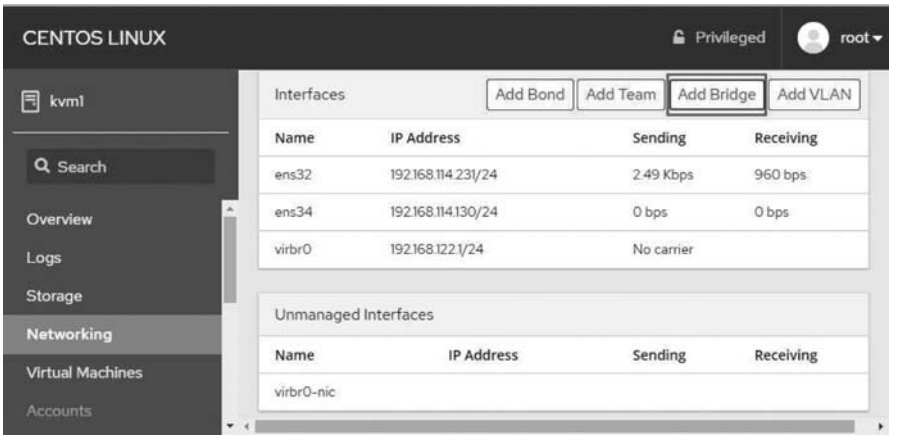

图 5-8　Cockpit 中的网络接口

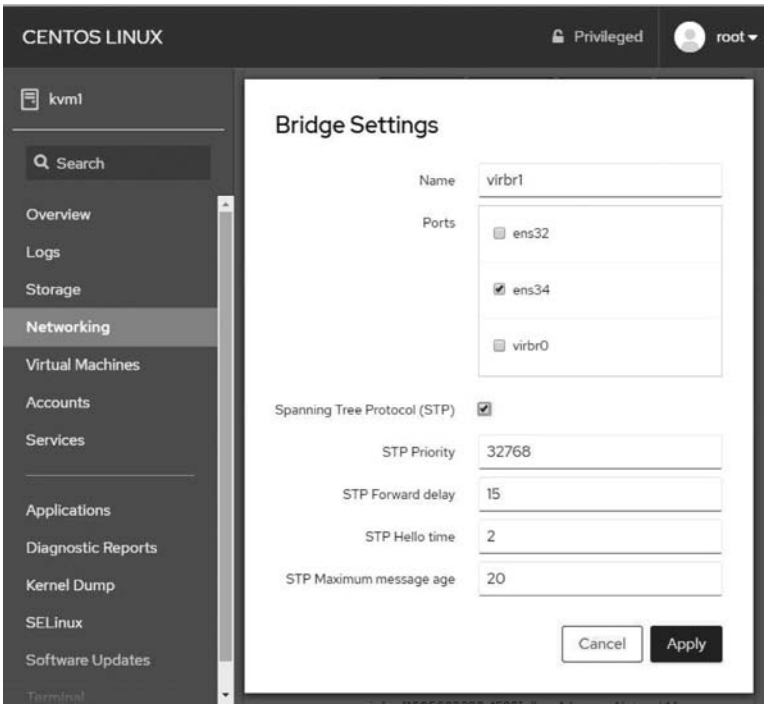

图 5-9　在 Cockpit 中创建新的网桥

可以在新页面中设置 IP 地址、STP 及子接口等信息，也可以单击 Delete 按钮删除网桥，如图 5-11 所示。

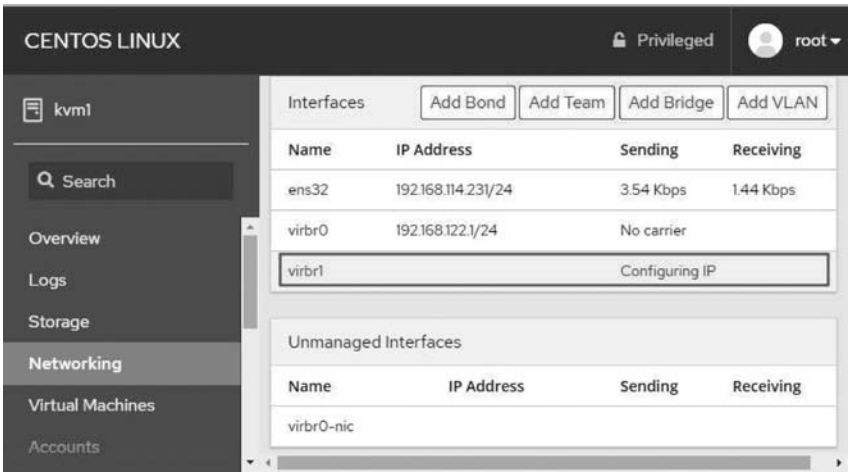

图 5-10　成功创建的新网桥

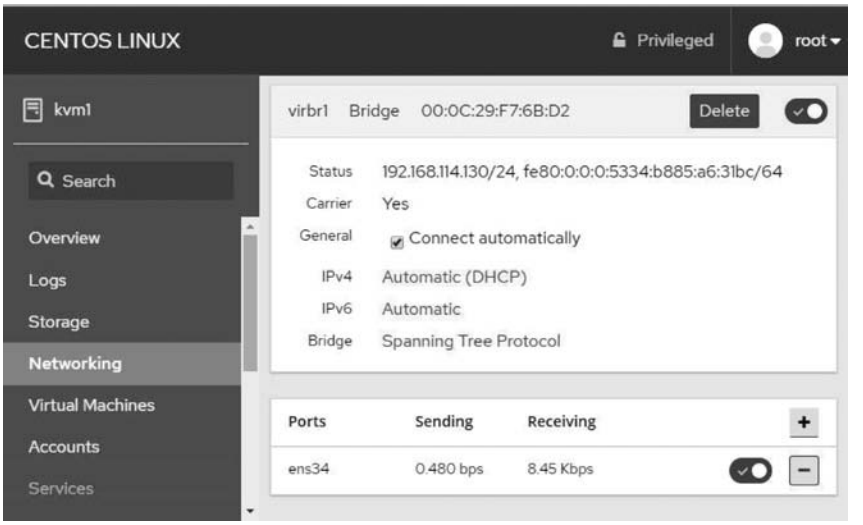

图 5-11　网桥的详细信息

5.4　KVM/libvirt 常用的网络类型

在掌握了 TUN/TAP 设备及网桥基本原理之后，我们就可以来学习 KVM/libvirt 常用的网络类型了。

本节先介绍虚拟机支持的网络、libvirt 管理的虚拟网络，然后讲解具体网络的工作原理。

5.4.1　虚拟机支持的网络

为虚拟机创建网络连接的时候，除了需要设置虚拟网卡的设备类型（如：virtio、e1000或rtl8139）之外，我们还需要为其指定网络连接的目标，可以将其分为两类。

（1）虚拟的网络：由libvirt来管理维护，例如NAT、Routed和Isolated等。虚拟机的配置是interface type＝"network"。

（2）共享的物理设备：由宿主机操作系统管理维护，例如带物理网卡子接口的网桥、宿主机的物理网卡等。虚拟机配置是interface type＝"bridge"或interface type＝"direct"。

在RHEL/CentOS 8中，不同管理工具支持的虚拟机的网络类型、表示方法有些细微的差异。例如：virt-manager（virt-manager-2.2.1-3.el8.noarch）使用Network source的一个选项来设置虚拟机的网络连接，如图5-12所示。Cockpit（cockpit-machines-211.3-1.el8.noarch）则使用Interface Type和Model两个选项，如图5-13所示，而通过virsh的edit子命令编辑虚拟机的XML文件则可以支持所有的网络类型（https://libvirt.org/formatdomain.html）。

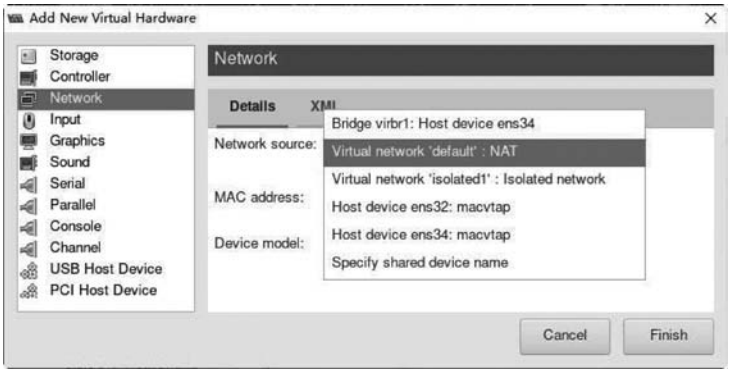

图5-12　virt-manager-2.2.1-3.el8使用Network Source的一个选项来配置虚拟机的网络接口

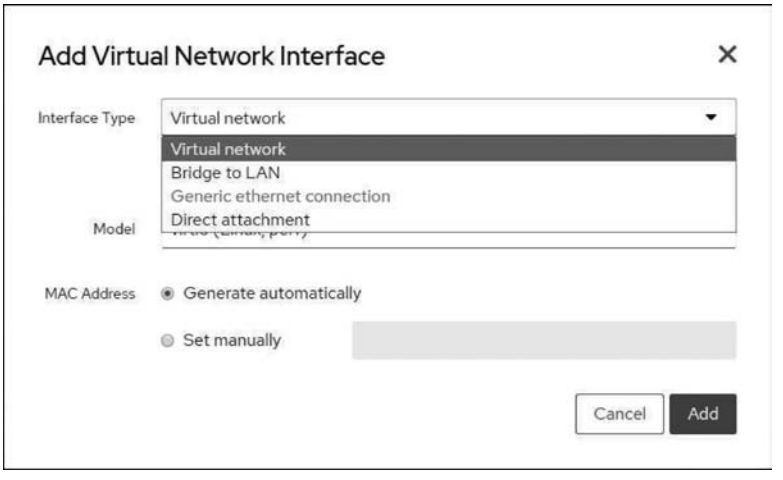

图5-13　cockpit-machines-211.3使用Interface Type和Model两个选项来配置虚拟机网络接口

表 5-1～表 5-8 是不同管理方式的对照。

表 5-1 连接到虚拟网络 default

工具	网卡配置
virt-manager	Virtual network 'default': NAT
Cockpit	Interface Type: Virtual Network Source: default
虚拟机 XML	<interface type="network"> <source network="default"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-2 连接到虚拟网络 isolated1

工具	网卡配置
virt-manager	Virtual Network 'isolated1': Isolated Network
Cockpit	Interface Type: Virtual Network Source:isolated1
虚拟机 XML	<interface type="network"> <source network="isolated1"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-3 连接到桥接网络 virbr1

工具	网卡配置
virt-manager	Bridge virbr1: Host device ens34
Cockpit	Interface Type: Virtual Network Source: virbr1
虚拟机 XML	<interface type="bridge"> <source bridge="virbr1"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-4 连接宿主机设备 ens34(VEPA)

工具	网卡配置
virt-manager	Host device ens34: macvtap, Source mode: VEPA
Cockpit	Interface Type: Direct attachment Source: ens34 默认是 vepa
虚拟机 XML	<interface type="direct"> <source dev="ens34" mode="vepa"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-5 连接宿主机设备 ens34(Bridge)

工具	网卡配置
virt-manager	Host device ens34：macvtap，Source mode：Bridge
Cockpit	Interface Type：Direct attachment Source：ens34 需要手工修改 mode
虚拟机 XML	<interface type="direct"> <source dev="ens34" mode="bridge"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-6 连接宿主机设备 ens34(Private)

工具	网卡配置
virt-manager	Host device ens34：macvtap，Source mode：Private
Cockpit	Interface Type：Direct attachment Source：ens34 需要手工修改 mode
虚拟机 XML	<interface type="direct"> <source dev="ens34" mode="private"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-7 连接宿主机设备 ens34(Passthrough)

工具	网卡配置
virt-manager	Host device ens34：macvtap，Source mode：Passthrough
Cockpit	Interface Type：Direct attachment Source：ens34 需要手工修改 mode
虚拟机 XML	<interface type="direct"> <source dev="ens34" mode="passthrough"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

表 5-8 连接指定共享设备

工具	网卡配置
virt-manager	Specify shared device name Bridge name：brtest
Cockpit	不支持
虚拟机 XML	<interface type="bridge"> <source bridge="brtest"/> <mac address="52:54:00:xx:xx:xx"/> <model type="virtio"/> </interface>

5.4.2 libvirt 管理的虚拟网络

在 RHEL/CentOS 8 中启动 libvirtd 守护程序时，libvirtd 会读取/etc/libvirt/qemu/networks/autostart/中的符号链接所指向的虚拟网络配置文件，然后使用 NetworkManager 来创建相应的网桥、TAP 等设备，并根据需要配置 IP 转发、路由表和 iptables 中的 NAT 规则，从而为虚拟机提供虚拟网络环境。

对于虚拟网络，RHEL/CentOS 8 中不同管理工具也有一些细微的差异。例如：virt-manager（virt-manager-2.2.1-3.el8.noarch）使用的术语是 Mode 和 Forward to，共用 5 种 Mode，如图 5-14 所示。Cockpit（cockpit-machines-211.3-1.el8.noarch）使用的术语主要是 Forward Mode 和 Device，仅有 3 种 Forward Mode，如图 5-15 所示，而通过 virsh 的 net-create、net-define、net-edit 和 net-update 子命令编辑网络 XML 文件则支持所有的虚拟网络类型（https://libvirt.org/formatnetwork.html）。

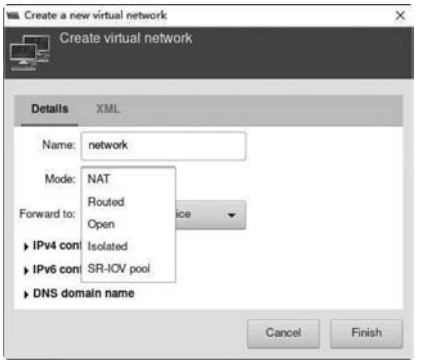

图 5-14 virt-manager-2.2.1-3.el8 创建虚拟网络

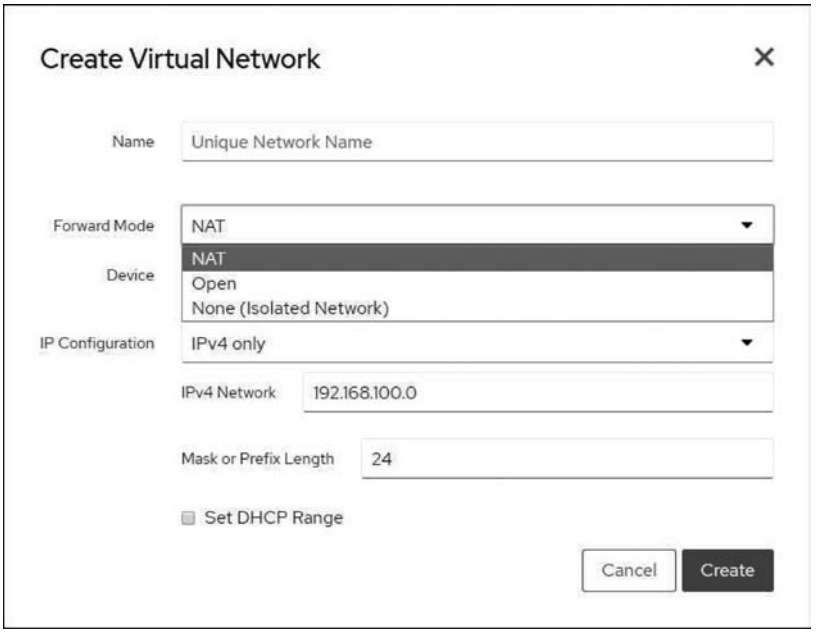

图 5-15 cockpit-machines-211.3 创建虚拟网络

表 5-9～表 5-13 是不同管理方式的对照。

表 5-9 NAT 模式虚拟网络

工具	虚拟网络配置
virt-manager	Mode：NAT，Forward to：Any physical device
Cockpit	Forward Mode：NAT，Device：Automatic
虚拟网络 XML	<network> <name>nat1</name> <forward mode="nat"/> <domain name="nat1"/> <ip address="192.168.100.1" netmask="255.255.255.0"> <dhcp> <range start="192.168.100.128" end="192.168.100.254"/> </dhcp> </ip> </network>

表 5-10 开放模式虚拟网络

工具	虚拟网络配置
virt-manager	Mode：Open
Cockpit	Forward Mode：Open，IP Configuration：IPv4 only
虚拟网络 XML	<network> <name>open1</name> <forward mode="open"/> <domain name="open1"/> <ip address="192.168.100.1" netmask="255.255.255.0"> <dhcp> <range start="192.168.100.128" end="192.168.100.254"/> </dhcp> </ip> </network>

表 5-11 隔离模式虚拟网络

工具	虚拟网络配置
virt-manager	Mode：Isolated
Cockpit	Forward Mode：None(Isolated Network)，IP Configuration：IPv4 only
虚拟网络 XML	<network> <name>isolated1</name> <domain name="isolated1"/> <ip address="192.168.100.1" netmask="255.255.255.0">

续表

工具	虚拟网络配置
虚拟网络 XML	<dhcp> <range start="192.168.100.128" end="192.168.100.254"/> </dhcp> </ip> </network>

表 5-12 路由模式虚拟网络

工具	虚拟网络配置
virt-manager	Mode：Routed，Forward to：Any physical device
Cockpit	不适用
虚拟网络 XML	<network> <name>route1</name> <forward mode="route"/> <domain name="route1"/> <ip address="192.168.100.1" netmask="255.255.255.0"> <dhcp> <range start="192.168.100.128" end="192.168.100.254"/> </dhcp> </ip> </network>

表 5-13 SR-IOV 模式虚拟网络

工具	虚拟网络配置
virt-manager	Mode：SR-IOV pool
Cockpit	不适用
虚拟网络 XML	<network> <name>sriov1</name> <forward mode="hostdev" managed="yes"/> </network>

5.4.3 NAT 模式

对于桌面虚拟化或测试环境来讲，NAT 模式是最常用的虚拟网络模式。不需要进行特别配置，此模式的虚拟机就可以访问外部网络，它还允许宿主机与虚拟机之间进行通信。NAT 模式的主要“缺点”是宿主机之外的系统无法访问虚拟机。默认的 default 网络就是 NAT 模式，如图 5-3 所示。

NAT 模式的虚拟网络是在 iptables 的帮助下创建的，其实是使用伪装选项，因此，停止

iptables 会导致虚拟机内部网络的中断。下面通过实验进行验证,代码如下:

```
1 # virsh domiflist centos6.10
  Interface    Type     Source     Model     MAC
  -------------------------------------------------------
  vnet0        network  default    virtio    52:54:00:32:f8:e2

2 # virsh domifaddr centos6.10
   Name      MAC address          Protocol       Address
  -------------------------------------------------------
   vnet0     52:54:00:32:f8:e2    ipv4           192.168.122.18/24

3 # ssh 192.168.122.18 "ping -c 1 www.baidu.com"
  root@192.168.122.18's password: 输入虚拟机操作系统 root 的密码
  PING www.wshifen.com (103.235.46.39) 56(84) Bytes of data.
  64 Bytes from 103.235.46.39: icmp_seq=1 ttl=127 time=418 ms

  --- www.wshifen.com ping statistics ---
  1 packets transmitted, 1 received, 0% packet loss, time 418ms
  rtt min/avg/max/mdev = 418.056/418.056/418.056/0.000 ms
```

第 1 行命令的输出结果显示虚拟机 centos6.10 有一个连接到 default 网络的网络接口。

第 2 行命令的输出结果显示这个接口的 IP 地址是 192.168.122.18/24。

第 3 行命令的输出结果验证了两个结论,一个是在宿主机可以访问处于 NAT 网络的虚拟机,另外一个是从这台虚拟机可以访问宿主机外部的网络。

接下来执行的命令如下:

```
4 # systemctl stop firewalld.service

5 # iptables -L -t nat
  Chain PREROUTING (policy ACCEPT)
  target     prot opt source               destination

  Chain INPUT (policy ACCEPT)
  target     prot opt source               destination

  Chain POSTROUTING (policy ACCEPT)
  target     prot opt source               destination

  Chain OUTPUT (policy ACCEPT)
  target     prot opt source               destination

6 # ssh 192.168.122.18 "ping -c 1 www.baidu.com"
  root@192.168.122.18's password:
```

```
PING www.wshifen.com (103.235.46.39) 56(84) Bytes of data.

--- www.wshifen.com ping statistics ---
1 packets transmitted, 0 received, 100 % packet loss, time 10000ms
```

第 4 行命令停止了防火墙服务。从第 5 行命令的输出中可以看出 iptables 的 NAT 表中的所有链的规则均为空。

从第 6 行命令的输出结果可以看出，这台虚拟机现在无法访问宿主机外部的网络。

提示：域名解析和 DHCP 服务是由 dnsmasq 进程提供的，停止防火墙服务不会影响此进程，所以第 6 行命令的输出显示域名解析服务工作正常。

5.4.4 桥接模式

libvirt 不能直接管理桥接模式的网络，所以需要先使用 NetworkManager、Cockpit 等工具在操作系统中创建一个网桥，然后将一个物理网卡（或捆绑在一起的多个物理网卡）分配给网桥作为子接口，最后将虚拟机连接到此网桥（虚拟交换机），如图 5-16 所示。

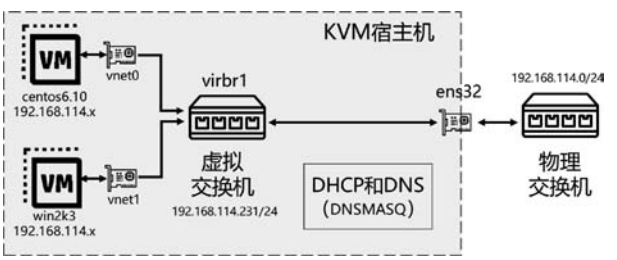

图 5-16 桥接模式的网络

网桥是在 OSI 网络模型的第 2 层上运行的。网桥（虚拟交换机）virbr1 通过 ens32 与宿主机外部的网络相连接。虚拟机 centos6.10、win2k3 与宿主机在同一个子网中，外部物理网络上的主机可以检测到它们并对其进行访问。

5.4.5 隔离模式

顾名思义，这是一种封闭的网络，连接在此虚拟交换机的虚拟机可以彼此通信，也可以与宿主机进行通信，但是它们的流量不会通过宿主机扩散到外部，当然也无法从宿主机外部访问它们，如图 5-17 所示。

5.4.6 路由模式

在路由模式下，宿主机充当路由器并配置路由规则，虚拟机通过虚拟交换机与物理网络相连，如图 5-18 所示。使用此模式有一个关键点：必须在外部的路由器或网关设备上设置正确

的 IP 路由，以便答复数据包返回宿主机，例如：需要有“将目标地址是 192.168.100.0/24 的数据包转发给 ens32 的 IP 地址”的路由，否则这些回复数据包将永远不会到达宿主机。

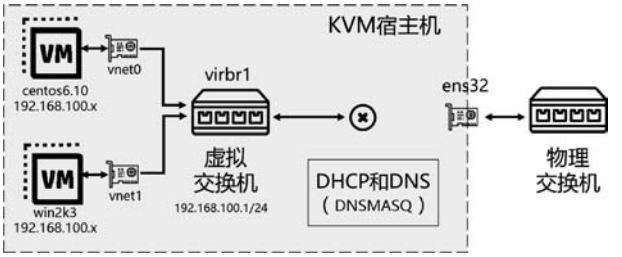

图 5-17　隔离模式的网络

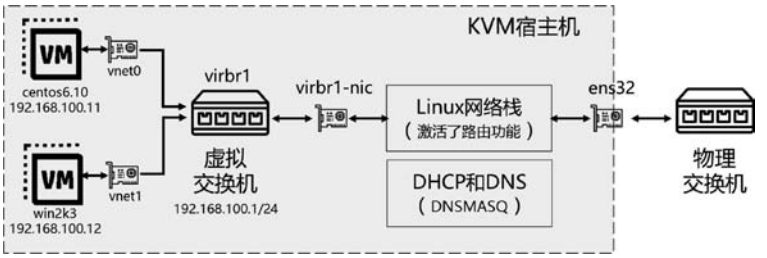

图 5-18　路由模式的网络

提示：除非有特殊的需求（如防火墙的 DMZ 区域），才会创建这种复杂性的网络，否则通常不会使用此模式。

5.4.7　开放模式

开放模式与路由模式类似，区别在于 libvirt 是否为虚拟网络设置防火墙规则。

在路由模式中，libvirt 会配置防火墙规则，例如允许 DHCP、DNS 等流量，而在配置开放模式时，libvirt 不会在此虚拟网络生成任何 iptables 规则，这样就需要用户自行配置。这就是“开放”所代表的含义。

5.4.8　直接附加模式

虚拟机的直接附加模式既没有使用 libvirt 管理的虚拟网络，也没有使用宿主机上的网桥，而是通过宿主机上的 macvtap 驱动程序，将虚拟机的网卡直接附加到宿主机上的指定物理网卡，如图 5-19 所示。

这种模式最大的优点是在物理交换机上可以同时知道宿主机和虚拟机的网卡，从而可以进行统一

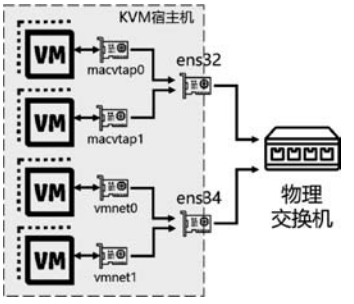

图 5-19　直接附加模式

管理。

直接附加有 4 种模式：VEPA、bridge、private、passthrough，它们在流量控制上有一些差异，默认为 VEPA 模式。

注意：不要将此模式与宿主机设备直通(passthrough)混淆。

5.4.9 PCI 直通与 SR-IOV

KVM Hypervisor 可以将宿主机上符合条件的 PCI 设备分配给虚拟机，这就是 PCI 直通(PCI passthrough)。它可以使虚拟机独占式地访问 PCI 设备，而且不需要或很少需要 KVM 的参与，所以可以提高性能。

目前大多数网卡支持这种特性，通过 PCI 直通将它们分配给虚拟机，从而满足对网络性能要求比较高的应用需求。

但是在服务器上安装 PCI 或 PCI-E 设备的数量总是有限的，而且随着设备数量的增加，也会增加成本。SR-IOV(Single Root Input/Output Virtualization)就是针对这种问题的一种解决方案。它可以将单个物理 PCI 设备划分为多个虚拟的 PCI 设备，然后把它们分配给虚拟机。例如，将宿主机上支持 SR-IOV 功能的网卡划分成多个独立的虚拟网卡，将每个虚拟网卡分配给一个虚拟机使用。

下面就以 Intel 的 X540-AT2 以太网卡为例进行说明。

首先，保证宿主机上有正确的驱动程序，可以识别到这块网卡，然后使用 virt-manager 为一台虚拟机添加硬件，单击 PCI Host Device，右边就会显示一系列可以分配给虚拟机的 PCI 设备。选中所需的 Intel 的 X540 的 Virtual Function，然后单击 Finish 按钮，如图 5-20 所示。这样虚拟机就可以使用这个 PCI 直通设备了。

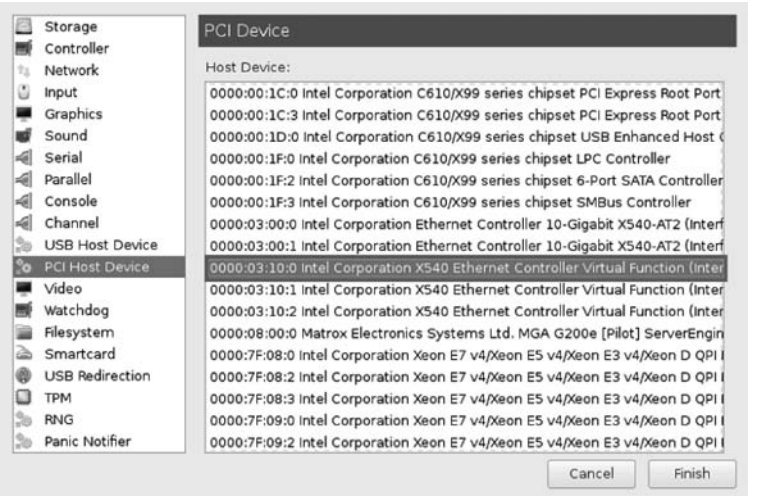

图 5-20　给虚拟机添加支持 SR-IOV 的网卡

5.5　创建和管理隔离的网络

隔离的网络是最简单的网络。本节我们将参考图 5-17 所示的拓扑，通过 virt-manager、Cockpit 和 virsh 来创建和管理一个名为 isolated1 的隔离网络。

5.5.1　通过 virt-manager 创建和管理隔离网络

首先，我们通过 virt-manager 进行创建，然后通过 virsh 及其他操作系统命令进行查看。

（1）在 virt-manager 的 Edit 菜单上，选择 Connection Details。

（2）在打开的 Connection Details 菜单中单击 Virtual Networks 选项卡。

（3）窗口的左侧列出了所有可用的虚拟网络。单击左下角的 + 按钮，会出现 Create a new virtual network 窗口，如图 5-21 所示。

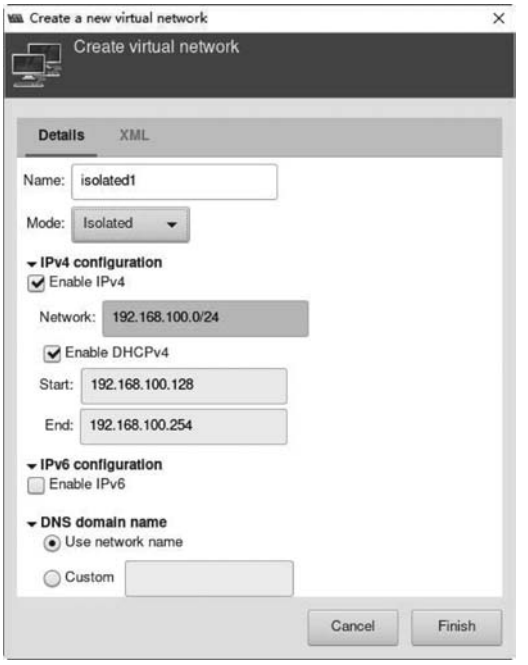

图 5-21　在 virt-manager 中创建新的隔离网络

（4）设置新虚拟网络的名称，将 Mode 设置为 Isolated，设置适合的 IPv4 configuration。

（5）单击 XML 标签，会看到虚拟网络的 XML 定义，如图 5-22 所示。

（6）单击 Finish 按钮完成新虚拟网络的创建。新的虚拟网络的信息如图 5-23 所示。

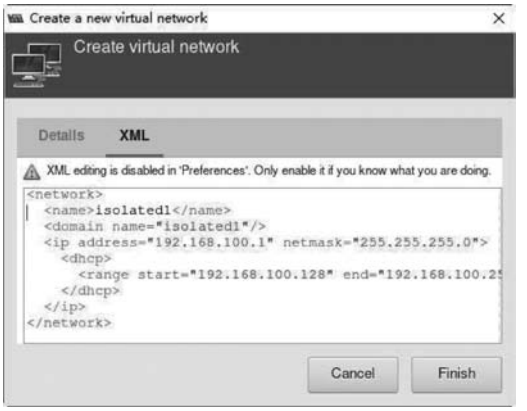

图 5-22　新的隔离网络的 XML 定义

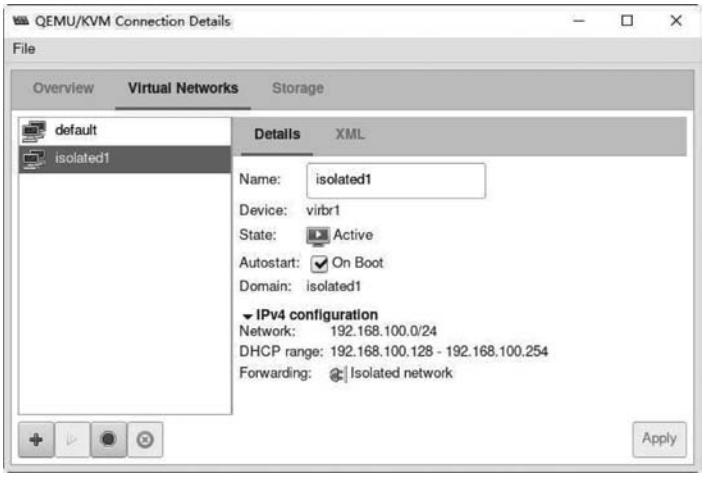

图 5-23　新的隔离网络的详细信息

下面通过 virsh 及其他操作系统命令来查看这个新的虚拟网络，命令如下：

```
1 # virsh net - list
   Name        State      Autostart       Persistent
  ----------------------------------------------------
   default     active     yes             yes
   isolated1   active     yes             yes

2 # virsh net - dumpxml isolated1
  < network >
    < name > isolated1 </name >
    < uuid > 44505edf - d08b - 4552 - a9f9 - 0254b80db693 </uuid >
    < bridge name = 'virbr1' stp = 'on' delay = '0'/>
```

```
    <mac address = '52:54:00:9a:14:dd'/>
    <domain name = 'isolated1'/>
    <ip address = '192.168.100.1' netmask = '255.255.255.0'>
      <dhcp>
        <range start = '192.168.100.128' end = '192.168.100.254'/>
      </dhcp>
    </ip>
  </network>
```

第 1 行命令的输出显示系统新增加了一个名为 isolated1 的虚拟网络，它是永久的、自动启动的，而且当前状态是已激活的。

第 2 行命令的输出显示了 isolated1 的详细信息，包括以下信息。

(1) name：提供了虚拟网络的名称。

(2) uuid：为虚拟网络提供了全局唯一的标识符。

(3) bridge：name 属性定义了将用于构建虚拟网络的网桥设备的名称。虚拟机将连接到该网桥，从而使它们可以相互通信。由于当前没有 forward 元素，所以这是一个隔离网络。以后会学到 mode 等于 nat、route 或 open 的 forward 元素。属性 stp 指定是否支持生成树协议。属性 delay 指定以秒为单位设置延迟值(默认为 0)。

(4) mac：address 属性定义了一个 MAC(硬件)地址。

(5) domain：name 属性定义了虚拟网络的名称。

(6) ip：address 定义了点分十进制格式的 IPv4 地址或标准冒号分隔的十六进制格式的 IPv6 地址，这些地址将在与虚拟网络关联的网桥设备上进行配置。netmask 定义了子网掩码。

(7) dhcp：指定在此虚拟网络上启用 DHCP 服务。

(8) range：start 和 end 属性指定了要提供给 DHCP 客户端的地址池的边界。

接下来执行的命令如下：

```
3 # ls /etc/libvirt/qemu/networks/
  autostart default.xml isolated1.xml

4 # cat /etc/libvirt/qemu/networks/isolated1.xml
  <!--
  WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
  OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
    virsh net-edit isolated1
  or other application using the libvirt API.
  -->

  <network>
    <name>isolated1</name>
```

```
    <uuid>44505edf-d08b-4552-a9f9-0254b80db693</uuid>
    <bridge name='virbr1' stp='on' delay='0'/>
    <mac address='52:54:00:9a:14:dd'/>
    <domain name='isolated1'/>
    <ip address='192.168.100.1' netmask='255.255.255.0'>
      <dhcp>
        <range start='192.168.100.128' end='192.168.100.254'/>
      </dhcp>
    </ip>
  </network>

5 # ls /etc/libvirt/qemu/networks/autostart/ -l
  total 0
  lrwxrwxrwx. 1 root root 14 Jul 30 16:30 default.xml -> ../default.xml
  lrwxrwxrwx. 1 root root 40 Nov 23 17:50 isolated1.xml -> /etc/libvirt/qemu/networks/
isolated1.xml
```

第3行命令的输出显示在/etc/libvirt/qemu/networks/目录中，新增了1个XML文件isolated1.xml。不建议直接编辑这个文件，而要使用virsh net-edit进行编辑，这是因为它在保存文件时会进行一些必要的检查。

第5行命令的输出显示在/etc/libvirt/qemu/networks/autostart/目录中，有一个指定配置文件的符号链接，这样可以保证在启动libvirtd守护程序时自动启动这个虚拟网络。

接下来执行的命令如下：

```
6 # ip address
  …
  8: virbr1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:9a:14:dd brd ff:ff:ff:ff:ff:ff
      inet 192.168.100.1/24 brd 192.168.100.255 scope global virbr1
         valid_lft forever preferred_lft forever
  9: virbr1-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr1 state DOWN
group default qlen 1000
      link/ether 52:54:00:9a:14:dd brd ff:ff:ff:ff:ff:ff

7 # nmcli connection
  NAME    UUID                                  TYPE       DEVICE
  ens32   152beb06-47c5-c5e8-95a9-385590654382  ethernet   ens32
  virbr0  2ffc46c3-9abb-4071-b5f8-61557313d623  bridge     virbr0
  virbr1  44d4f380-9fde-4d84-a8c9-1a24d159e30b  bridge     virbr1

8 # nmcli device
  DEVICE       TYPE       STATE       CONNECTION
  ens32        ethernet   connected   ens32
```

```
virbr0          bridge          connected          virbr0
virbr1          bridge          connected          virbr1
lo              loopback        unmanaged          --
virbr0-nic      tun             unmanaged          --
virbr1-nic      tun             unmanaged          --
```

第6行命令的输出显示系统新增加了1个名为virbr1的网桥，它的IP地址是我们指定的192.168.100.1/24。此网桥有一个名为virbr1-nic的子接口，virbr1与virbr1-nic的MAC地址都是52:54:00:9a:14:dd。

在第7行、第8行的nmcli的子命令的输出中，也会看到virbr1与virbr1-nic。

```
9 # route -n
    Kernel IP routing table
  Destination      Gateway          Genmask          Flags  Metric  Ref  Use  Iface
  0.0.0.0          192.168.114.2    0.0.0.0          UG     102     0    0    ens32
  192.168.100.0    0.0.0.0          255.255.255.0    U      0       0    0    virbr1
  192.168.114.0    0.0.0.0          255.255.255.0    U      102     0    0    ens32
  192.168.122.0    0.0.0.0          255.255.255.0    U      0       0    0    virbr0
```

第9行命令的输出显示在宿主机的路由表中，新增了一个到192.168.100.0/24的路由，接口是virbr1交换机，所以可以在宿主机上访问连接到isolated1网络的虚拟机。

提示：在创建隔离模式的虚拟网络时，不会引起iptables规则的变化。

当不再需要此虚拟网络的时候，可以在virt-manager中先单击按钮停止网络，然后单击按钮删除网络。

5.5.2　通过Cockpit创建和管理隔离网络

通过Cockpit创建和管理隔离网络与virt-manager很类似。

(1) 单击左边导航中的Virtual Machines链接，再单击Networks链接。

(2) 单击Create Virtual Network链接，然后为新虚拟网络设置属性，然后单击Create按钮以完成创建，如图5-24所示。需要注意的是：IPv4 Network选项设置的是新创建网桥的IP地址，而不是网段的IP地址。

(3) 与virt-manager不同，还需要单击Activate按钮以激活新创建的虚拟网络。还可以根据需要选中Run when host boots检查框，将虚拟网络设置为自动启动，如图5-25所示。

当不再需要此虚拟网络的时候，可以单击Deactive按钮停止网络，然后单击Delete按钮删除网络。

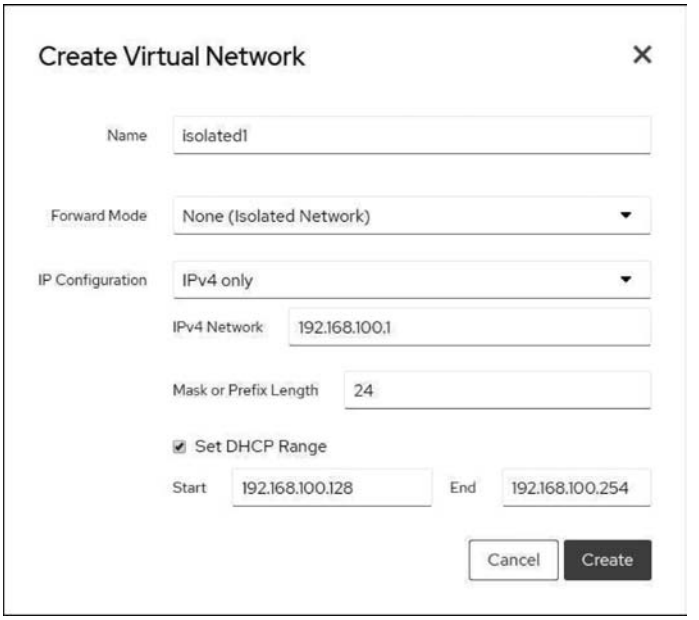

图 5-24　在 Cockpit 中创建新的隔离网络

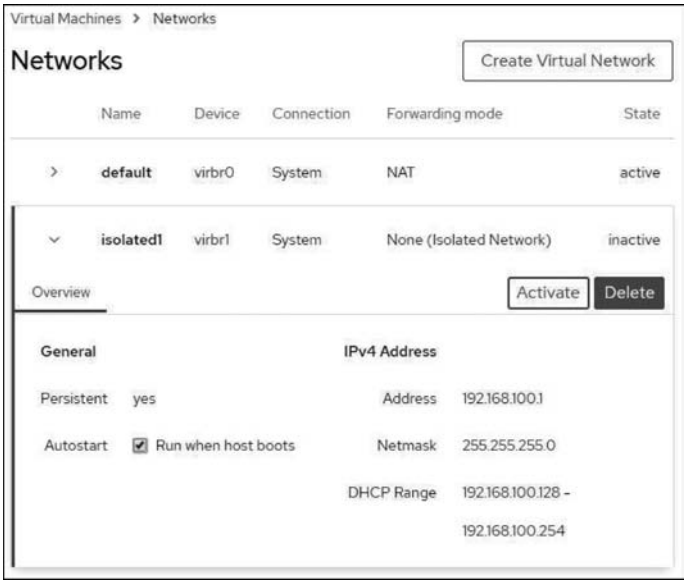

图 5-25　新的隔离网络的详细信息

5.5.3　通过 virsh 创建和管理隔离网络

virsh 管理虚拟网络的子命令多数是以 net 开头的，可以通过 virsh help network 获得

这些子命令的清单，通过 virsh help 获得某个子命令的详细帮助，命令如下：

```
1 # virsh help network
  Networking (help keyword 'network'):
    net-autostart     autostart a network
    net-create        create a network from an XML file
    net-define        define an inactive persistent virtual network or modify an existing
persistent one from an XML file
    net-destroy       destroy (stop) a network
    net-dhcp-leases   print lease info for a given network
    net-dumpxml       network information in XML
    net-edit          edit XML configuration for a network
    net-event         Network Events
    net-info          network information
    net-list          list networks
    net-name          convert a network UUID to network name
    net-start         start a (previously defined) inactive network
    net-undefine      undefine a persistent network
    net-update        update parts of an existing network's configuration
    net-uuid          convert a network name to network UUID

2 # virsh help net-create
  NAME
    net-create - create a network from an XML file
  SYNOPSIS
      net-create <file>
  DESCRIPTION
    Create a network.
  OPTIONS
    [--file] <string> file containing an XML network description
```

virsh 的 net-create 子命令可以根据 1 个 XML 文件中的设置来创建临时性的(transient)虚拟网络。net-define 子命令既可以创建新的虚拟网络，也可以修改现有的虚拟网络。net-define 子命令创建的新的虚拟网络是持久的，但默认为未激活，需要使用 net-start 子命令进行激活。

下面，我们使用 net-define 子命令来做一个实验，命令如下：

```
3 # vi isolated1.xml
  #新虚拟网络的定义如下
  <network>
    <name>isolated1</name>
    <domain name="isolated1"/>
    <ip address="192.168.100.1" netmask="255.255.255.0">
      <dhcp>
        <range start="192.168.100.128" end="192.168.100.254"/>
      </dhcp>
```

```
    </ip>
  </network>

4 # virsh net-define isolated1.xml
  Network isolated1 defined from isolated1.xml

5 # virsh net-dumpxml isolated1
  <network>
    <name>isolated1</name>
    <uuid>744c4ba9-cb35-4acc-b577-00b75d4eee52</uuid>
    <bridge name='virbr1' stp='on' delay='0'/>
    <mac address='52:54:00:28:05:b0'/>
    <domain name='isolated1'/>
    <ip address='192.168.100.1' netmask='255.255.255.0'>
      <dhcp>
        <range start='192.168.100.128' end='192.168.100.254'/>
      </dhcp>
    </ip>
  </network>
```

第3行命令在当前目录下创建了一个新文件,在其中为新的虚拟网络设置了一些必要的属性。如果不指定模式,则默认为隔离模式。如果不指定 bridge 属性,则 libvirt 会根据默认值生成一个名称为 virbr 开头的网桥。

第4行命令定义了一个新的网桥。从第5行命令的输出中可以看到 libvirt 会生成一些属性值,包括 uuid、bridge、mac 等。

接下来执行的命令如下:

```
6 # virsh net-list
  Name       State      Autostart      Persistent
  ----------------------------------------------------
  default    active     yes            yes

7 # virsh net-list --all
  Name       State      Autostart      Persistent
  ----------------------------------------------------
  default    active     yes            yes
  isolated1  inactive   no             yes

8 # virsh net-start isolated1
  Network isolated1 started

9 # virsh net-autostart isolated1
  Network isolated1 marked as autostarted
```

```
10 #virsh net-list --all
   Name       State       Autostart       Persistent
   ----------------------------------------------------
   default    active      yes             yes
   isolated1  active      yes             yes
```

由于新增加的虚拟网络并没有被激活，所以它不会出现在第 6 行命令的输出中。第 7 行命令有--all 选项，所以可以看到这个新网络。

第 8 行命令启动这个网络。第 9 行命令将其设置为自动启动。

如果需要修改虚拟网络，则既可以通过 net-edit 子命令来编辑 XML 文件，也可以通过 net-update 子命令根据另外一个 XML 的文件进行更新。

当不再需要某个虚拟网络的时候，可以先使用 net-destroy 子命令停止它，然后使用 net-undefine 子命令删除它的定义。

5.5.4　使用隔离网络

隔离模式允许虚拟机之间相互通信，但是它们无法与外部物理网络进行通信。下面我们通过实验进行验证。

我们将虚拟机 centos6.10 和 win2k3 都更改到 isolated1 这个虚拟网络中。更改虚拟机网络连接的方法很简单，例如在 virt-manager 中更改虚拟机的网络配置即可，如图 5-26 所示。

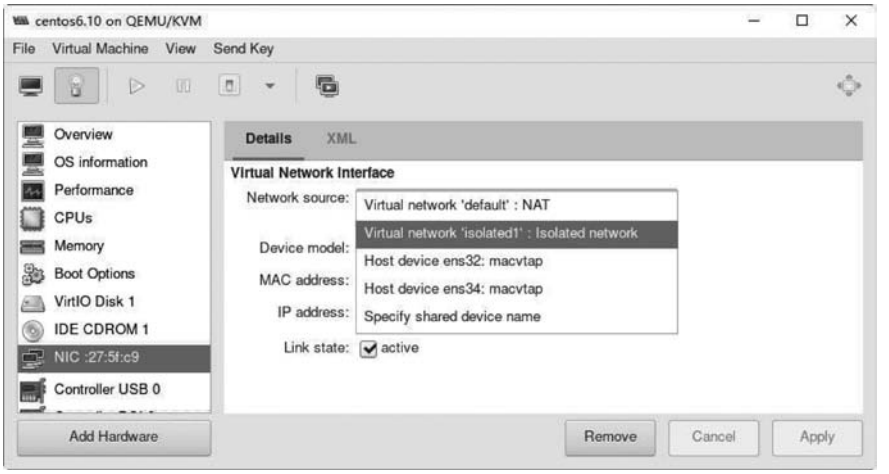

图 5-26　更新虚拟机的网络配置

使用 virsh 的 edit 可以修改虚拟机的 XML 文件。例如将原有虚拟网卡配置进行修改，原配置如下：

```
<interface type='network'>
  <mac address='52:54:00:27:5f:c9'/>
  <source network='default'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

修改后的配置如下：

```
<interface type='network'>
  <mac address='52:54:00:27:5f:c9'/>
  <source network='isolated1'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

最主要是修改<source network='isolated1'/>。修改之后重新启动虚拟机即可生效。如果想修改正在运行的虚拟机网卡配置，则需要使用 update-device 子命令，命令如下：

```
1 #vi new.xml
  <interface type='network'>
  <mac address='52:54:00:32:f8:e2'/>
  <source network='isolated1'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>

2 #virsh update-device --domain centos6.10 --file new.xml --persistent --live
  Device updated successfully

  #virsh edit centos6.10

3 #virsh domiflist centos6.10
  Interface     Type      Source      Model      MAC
  -------------------------------------------------------------
  vnet0         network   isolated1   virtio     52:54:00:32:f8:e2
```

第 1 行命令创建了包括新配置的 XML 文件。与原有 XML 文件的区别主要是<source network='isolated1'/>这一行。

第 2 行命令使用 XML 文件来更新设备。通过--domain 指定虚拟机的名称、ID 或 uuid，--file 后可指定 XML 文件，--persistent 使变化持久，--live 用于修改正在运行的虚拟机。

从第 3 行命令的输出可以看出虚拟机的网卡切换到 isolated1 虚拟网络了。

提示：如果虚拟机 IP 是动态 IP 地址，则需要等到 DHCP 租期过期后才会更新。如果需要立即更新，就需要在虚拟机操作系统中进行 DHCP 刷新操作了。

思考：一台宿主机上可以拥有多个隔离的网络，属于不同隔离网络的虚拟机是否可能通信呢？如果不可以，则是否可以通过调整配置来使它们之间相互通信呢？

5.6　创建和管理 NAT 的网络

5.6.1　使用多种方式创建 NAT 网络

在 RHEL/CentOS 8 上安装 KVM 虚拟化组件的时候，会自动创建一个名为 default 的 NAT 网络。一台宿主机上可以有多个采用 NAT 模式的虚拟网络，下面我们再创建一个名为 nat2 的 NAT 模式的虚拟网络。

如果使用 virt-manager 来创建，则需要将 Mode 设置为 NAT，可以保持 Forward to 为 Any physical device，设置适合的 IPv4 configuration，如图 5-27 所示。

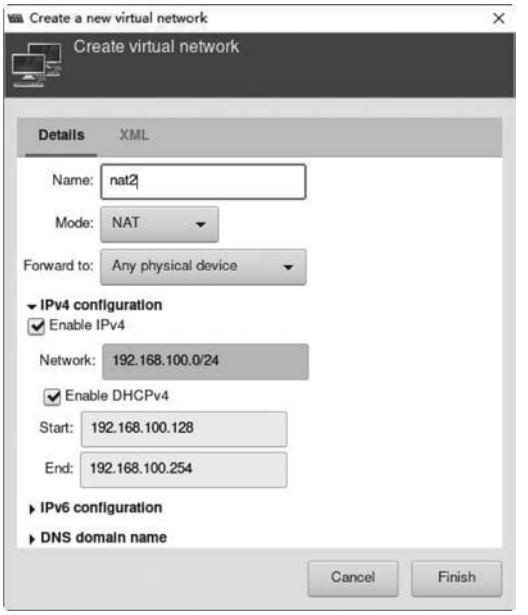

图 5-27　在 virt-manager 中创建新的 NAT 网络

单击 XML 选项卡可以查看新网络的 XML 定义，单击 Finish 按钮完成定义，如图 5-28 所示。

在 Cockpit 创建新的 NAT 网络的方式与 virt-manager 类似，不过需要注意的是：IPv4 Network 选项设置的是新创建网桥的 IP 地址，而不是网段的 IP 地址，如图 5-29 所示。

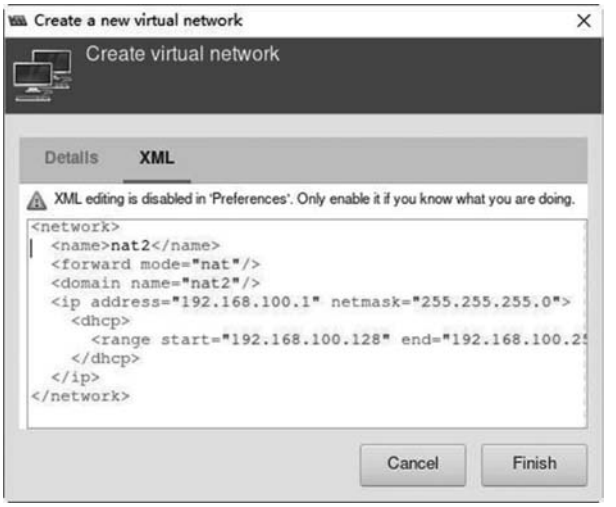

图 5-28　在 virt-manager 查看新的网络的 XML 定义

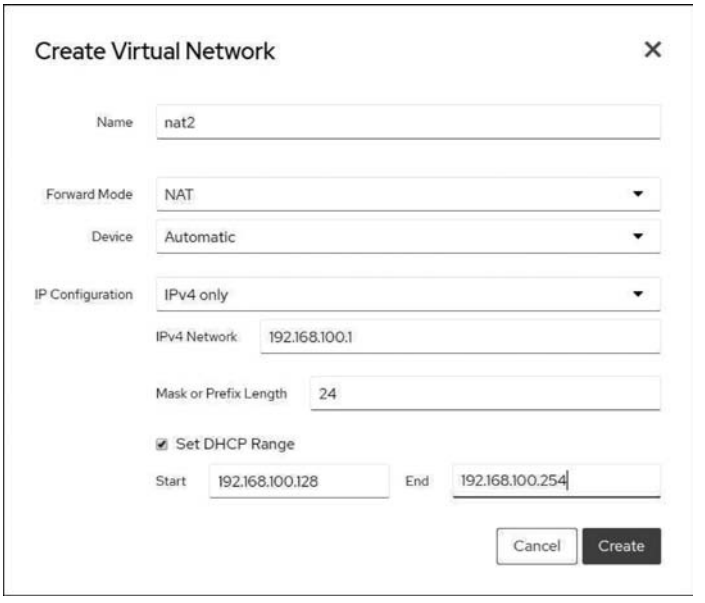

图 5-29　在 Cockpit 中创建新的 NAT 网络

通过 virsh 来创建 NAT 模式的网络，仍然需要使用 XML 格式的配置文件，命令如下：

```
1 # vi new.xml
  < network >
    < name > nat2 </name >
    < forward mode = "nat"/>
```

```
    <domain name = "nat2"/>
    <ip address = "192.168.100.1" netmask = "255.255.255.0">
      <dhcp>
        <range start = "192.168.100.128" end = "192.168.100.254"/>
      </dhcp>
    </ip>
  </network>

2 # virsh net - define new.xml
  Network nat2 defined from new.xml

3 # virsh net - autostart nat2
  Network nat2 marked as autostarted

4 # virsh net - start nat2
  Network nat2 started

5 # virsh net - list -- all
   Name       State     Autostart     Persistent
  ----------------------------------------------------------
   default    active    yes           yes
   nat2       active    yes           yes
```

第 1 行命令在当前目录下创建了一个新文件，在其中为新的虚拟网络设置了一些必要的属性。“<forward mode="nat"/>”定义的虚拟网络是 NAT。如果不指定 bridge 属性，则 libvirt 会生成一个以 virbr 开头的网桥。

第 2 行命令定义了一个新的网桥。

第 3 行命令设置为自动启动，第 4 行命令用于启动这个网络。

接下来执行的命令如下：

```
6 # ip address
  …
  6: virbr1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
  link/ether 52:54:00:4d:94:67 brd ff:ff:ff:ff:ff:ff
  inet 192.168.100.1/24 brd 192.168.100.255 scope global virbr1
  valid_lft forever preferred_lft forever
  7: virbr1-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr1 state DOWN
group default qlen 1000
  link/ether 52:54:00:4d:94:67 brd ff:ff:ff:ff:ff:ff

7 # virsh net - dumpxml nat2
  <network>
    <name>nat2</name>
```

```
  <uuid>8a54c7f3-e910-4d12-8312-34a11467ac86</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr1' stp='on' delay='0'/>
  <mac address='52:54:00:4d:94:67'/>
  <domain name='nat2'/>
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254'/>
    </dhcp>
  </ip>
</network>
```

第 6 行命令的输出显示：新增 1 个名为 virbr1 的网桥，它有 1 个名为 virbr1-nic 的子接口。网桥 virbr1 与 virbr1-nic 的 MAC 地址相同。

第 7 行命令会输出新网桥的详细信息，会看到有些属性使用了默认值或是自动生成的。

接下来执行的命令如下：

```
8 # iptables -L -t nat -n
  Chain PREROUTING (policy ACCEPT)
  target     prot opt source               destination

  Chain INPUT (policy ACCEPT)
  target     prot opt source               destination

  Chain POSTROUTING (policy ACCEPT)
  target     prot opt source               destination
  RETURN     all  --  192.168.100.0/24     224.0.0.0/24
  RETURN     all  --  192.168.100.0/24     255.255.255.255
  MASQUERADE tcp  --  192.168.100.0/24     !192.168.100.0/24     masq ports: 1024-65535
  MASQUERADE udp  --  192.168.100.0/24     !192.168.100.0/24     masq ports: 1024-65535
  MASQUERADE all  --  192.168.100.0/24     !192.168.100.0/24
  RETURN     all  --  192.168.122.0/24     224.0.0.0/24
  RETURN     all  --  192.168.122.0/24     255.255.255.255
  MASQUERADE tcp  --  192.168.122.0/24     !192.168.122.0/24     masq ports: 1024-65535
  MASQUERADE udp  --  192.168.122.0/24     !192.168.122.0/24     masq ports: 1024-65535
  MASQUERADE all  --  192.168.122.0/24     !192.168.122.0/24

  Chain OUTPUT (policy ACCEPT)
  target     prot opt source               destination
```

```
9 #route -n
  Kernel IP routing table
  Destination     Gateway         Genmask         Flags  Metric  Ref   Use  Iface
  0.0.0.0         192.168.114.2   0.0.0.0         UG     100     0     0    ns32
  192.168.100.0   0.0.0.0         255.255.255.0   U      0       0     0    virbr1
  192.168.114.0   0.0.0.0         255.255.255.0   U      100     0     0    ens32
  192.168.122.0   0.0.0.0         255.255.255.0   U      0       0     0    virbr0
```

libvirt 会根据 NAT 模式的虚拟网络的配置来修改 iptables 的 NAT 规则和系统的路由。

从第 8 行命令的输出可以看到在 iptables 的 NAT 表的 POSTROUTING 链中有多条针对 192.168.100.0/24 的规则。正是由于这些规则才使连接到 nat2 网络中的虚拟机可以访问外部的网络。

第 9 行命令的输出显示在宿主机的路由表中，新增了一个到 192.168.100.0/24 的路由，目标地址是这个网络的数据包将通过 virbr1 交换机输送出去。这样，就可以在宿主机上访问连接到 nat2 网络的虚拟机了，而且属于 default 网络的虚拟机也可以访问 nat2 网络的虚拟机。

5.6.2 使用 NAT 网络

在实验环境中，准备使用两台虚拟机来做实验。win2k3 还是属于 default 网络，将 centos6.10 修改为 nat2，命令如下：

```
1 #virsh edit centos6.10
  …
  <interface type='network'>
    <mac address='52:54:00:32:f8:e2'/>
    <source network='nat2'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  …

2 #virsh start centos6.10

3 #virsh start win2k3
```

第 1 行命令用于编辑虚拟配置的 XML 文件，修改网卡的属性 source，并将其修改为 <source network='nat2'/>。

第 2 行、第 3 行命令分别启动了这 2 台属于不同虚拟网络的虚拟机。

接下来执行的命令如下：

```
4 # ip address
    …
  8: vnet0: < BROADCAST,MULTICAST,UP,LOWER_UP > mtu 1500 qdisc fq_codel master virbr1 state
UNKNOWN group default qlen 1000
  link/ether fe:54:00:27:5f:c9 brd ff:ff:ff:ff:ff:ff
  inet6 fe80::fc54:ff:fe27:5fc9/64 scope link
  valid_lft forever preferred_lft forever
  9: vnet1: < BROADCAST,MULTICAST,UP,LOWER_UP > mtu 1500 qdisc fq_codel master virbr0 state
UNKNOWN group default qlen 1000
  link/ether fe:54:00:9d:57:9c brd ff:ff:ff:ff:ff:ff
  inet6 fe80::fc54:ff:fe9d:579c/64 scope link
  valid_lft forever preferred_lft forever
    …

5 # virsh domiflist centos6.10
  Interface    Type       Source     Model     MAC
  -------------------------------------------------------
  vnet0        network    nat2       virtio    52:54:00:27:5f:c9

6 # virsh domifaddr centos6.10
   Name     MAC address          Protocol     Address
   ------------------------------------------------------------------------
   vnet0    52:54:00:27:5f:c9    ipv4         192.168.100.139/24

7 # virsh domiflist win2k3
  Interface    Type       Source     Model     MAC
  -------------------------------------------------------
  vnet1        network    default    e1000     52:54:00:9d:57:9c

8 # virsh domifaddr win2k3
   Name     MAC address          Protocol     Address
   ------------------------------------------------------------------------
   vnet1    52:54:00:9d:57:9c    ipv4         192.168.122.41/24
```

从第 3 行命令的输出可以看到宿主机上新增了 2 个 TAP 类型的虚拟网络接口 vnet0 和 vnet1，它们分别是网桥 virbr1 和 virbr0 的子接口。

从第 5 行至第 8 行的输出可以看出：虚拟机 centos6.10 连接到虚拟网络 nat2，它的 IP 地址是 DNSMASQ 分配的 192.168.100.139/2。虚拟机 win2k3 连接到虚拟网络 default，它的 IP 地址是 DNSMASQ 分配的 192.168.122.41/24。

提示：宿主机上名称为 vnet 开头的 TAP 设备与虚拟机的以太网卡是“一对”设备，它们的 MAC 地址后面 5 组 2 位十六进制数字是相同的，例如：vnet0 与 centos6.10 的 eth0 的 MAC 地址都是以“54:00:27:5f:c9”结束。

下面，我们连接到虚拟机 centos6.10(192.168.100.139)来做一些测试，命令如下：

```
9 # ssh 192.168.100.139 "ping -c 2 www.baidu.com"
  root@192.168.100.139's password:
  PING www.wshifen.com (103.235.46.39) 56(84) Bytes of data.
  64 Bytes from 103.235.46.39: icmp_seq=1 ttl=127 time=278 ms
  64 Bytes from 103.235.46.39: icmp_seq=2 ttl=127 time=277 ms

  --- www.wshifen.com ping statistics ---
  2 packets transmitted, 2 received, 0% packet loss, time 1281ms
  rtt min/avg/max/mdev = 277.802/278.094/278.387/0.603 ms

10 # ssh 192.168.100.139 "ping -c 2 192.168.122.41"
   root@192.168.100.139's password:
   PING 192.168.122.41 (192.168.122.41) 56(84) Bytes of data.
   64 Bytes from 192.168.122.41: icmp_seq=1 ttl=127 time=1.96 ms
   64 Bytes from 192.168.122.41: icmp_seq=2 ttl=127 time=2.10 ms

   --- 192.168.122.41 ping statistics ---
   2 packets transmitted, 2 received, 0% packet loss, time 1005ms
   rtt min/avg/max/mdev = 1.962/2.034/2.107/0.085 ms
```

从第 9 行命令的输出中可以看出：从宿主机可以访问连接到虚拟网络 nat2 的虚拟机，而这些虚拟机可以通过宿主机的 NAT 功能访问外部网络资源。

从第 10 行命令的输出可以看出：连接到虚拟网络 nat2 的虚拟机也可以访问同一宿主机上的其他 NAT 网络中的虚拟机(default 网络中的 win2k3)。

5.7　创建和管理桥接的网络

桥接模式的网络是很常见的，外部网络上的主机可以访问这种连接模式网桥中的虚拟机。

下面按图 5-16 所示的拓扑来做一个桥接模式网络的实验。

5.7.1　在宿主机上创建网桥

执行的命令如下：

```
1 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
         valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
```

```
        valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff
      inet 192.168.114.231/24 brd 192.168.114.255 scope global noprefixroute ens32
        valid_lft forever preferred_lft forever
      inet6 fe80::c589:de19:7895:d32e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
      inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
        valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
      link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
```

第 1 行命令会显示创建新网桥之前网络接口的信息。IP 地址 192.168.114.231/24 是与 ens32 相关联的。

使用 Cockpit 来创建网桥是最简单的方法。新网桥的名称是 virbr1。将宿主机上唯一的物理网卡 en32 作为它的子接口，如图 5-30 所示。

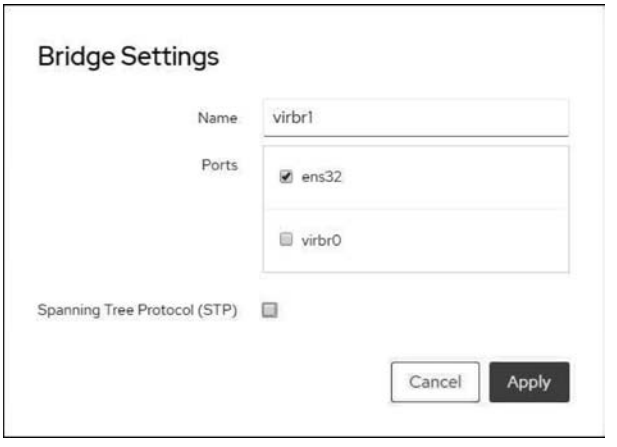

图 5-30　使用 Cockpit 创建新的网桥

接下来执行的命令如下：

```
2 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
```

```
          valid_lft forever preferred_lft forever
       inet6 ::1/128 scope host
          valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master virbr1 state UP
group default qlen 1000
       link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
       link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
       inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
          valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
       link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
  5: virbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
       link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff
       inet 192.168.114.231/24 brd 192.168.114.255 scope global noprefixroute virbr1
          valid_lft forever preferred_lft forever
       inet6 fe80::17b3:2554:31a6:df6e/64 scope link noprefixroute
          valid_lft forever preferred_lft forever

3 # nmcli connection
  NAME        UUID                                   TYPE       DEVICE
  virbr1      66629af3-7afb-4cbd-9170-020a9f9cb805   bridge     virbr1
  virbr0      665697d3-824e-4832-b057-14acf8d379e0   bridge     virbr0
  ens32       152beb06-47c5-c5e8-95a9-385590654382   ethernet   ens32
```

第 2 行命令输出了创建新网桥之后网络接口的信息。IP 地址 192.168.114.231/24 与 virbr1 相关联，ens32 则成为 virbr1 的子接口。网桥 virbr1 与第 1 个子接口也就是 ens32 的 MAC 地址是相同的，都是“00:0c:29:b3:73:36”。

注意：如果宿主机上仅有一个网卡，而且在远程通过 nmcli 或网络接口配置文件来创建网桥，就要特别小心，因为错误的配置会导致无法再进行远程连接。

接下来执行的命令如下：

```
4 # cat /etc/sysconfig/network-scripts/ifcfg-virbr1
  STP=no
  TYPE=Bridge
  PROXY_METHOD=none
  BROWSER_ONLY=no
  BOOTPROTO=none
  IPADDR=192.168.114.231
```

```
PREFIX = 24
GATEWAY = 192.168.114.2
DNS1 = 8.8.8.8
DEFROUTE = yes
IPV4_FAILURE_FATAL = no
IPV6INIT = yes
IPV6_AUTOCONF = yes
IPV6_DEFROUTE = yes
IPV6_FAILURE_FATAL = no
IPV6_ADDR_GEN_MODE = stable - privacy
NAME = virbr1
UUID = 66629af3 - 7afb - 4cbd - 9170 - 020a9f9cb805
DEVICE = virbr1
ONBOOT = yes
AUTOCONNECT_SLAVES = yes

5 # cat /etc/sysconfig/network - scripts/ifcfg - ens32
TYPE = Ethernet
PROXY_METHOD = none
BROWSER_ONLY = no
DEFROUTE = yes
IPV4_FAILURE_FATAL = no
NAME = ens32
DEVICE = ens32
ONBOOT = yes
IPADDR = 192.168.114.231
PREFIX = 24
GATEWAY = 192.168.114.2
DNS1 = 8.8.8.8
UUID = 152beb06 - 47c5 - c5e8 - 95a9 - 385590654382
BRIDGE = virbr1
```

第 4 行命令会显示新网桥 virbr1 的网络接口配置文件的内容，其中最重要的属性是 TYPE＝Bridge。

第 5 行命令显示了配置文件 ifcfg-ens32 的变化。这个修改后的配置文件不是特别“干净”，有一些无用信息。不过最关键的属性是 BRIDGE＝virbr1。

5.7.2 使用网桥

以虚拟机 centos6.10 为例，我们通过 Cockpit 来修改其网络接口的配置。在 Interface Type 中选择 Bridge to LAN，然后在 Source 中选择 virbr1，单击 Save 按钮保存更改，如图 5-31 所示。

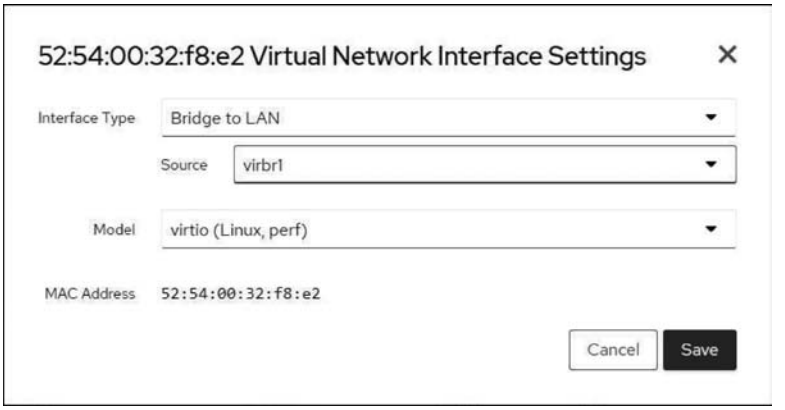

图 5-31　使用 Cockpit 更新虚拟机的网络配置

执行的命令如下：

```
1 # virsh domiflist centos6.10
  Interface     Type     Source     Model     MAC
  -------------------------------------------------------------
  -             bridge   virbr1     virtio    52:54:00:32:f8:e2

2 # virsh dumpxml centos6.10
  ...
  <interface type='bridge'>
    <mac address='52:54:00:32:f8:e2'/>
    <source bridge='virbr1'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  ...

3 # virsh start centos6.10

4 # ping -c 2 192.168.114.129
  PING 192.168.114.129 (192.168.114.129) 56(84) Bytes of data.
  64 Bytes from 192.168.114.129: icmp_seq=1 ttl=64 time=1.15 ms
  64 Bytes from 192.168.114.129: icmp_seq=2 ttl=64 time=0.940 ms

  --- 192.168.114.129 ping statistics ---
  2 packets transmitted, 2 received, 0% packet loss, time 3ms
  rtt min/avg/max/mdev = 0.940/1.047/1.154/0.107 ms
```

第 1 行命令的输出内容显示了此虚拟机已经连接网桥 virbr1。

第 2 行命令的输出内容显示了它的网桥接口类型是< interface type＝'bridge'>。

第 3 行命令用于启动虚拟机。这时，虚拟机 centos6.10 会从宿主机所在物理网络中的 DHCP 服务器获得 IP 地址。我们可以登录到此虚拟机查看其 IP 地址，在本示例中是 192.168.114.129

第 4 行命令用于对这个 IP 地址进行测试。当然外部物理网络上的其他主机也可以访问此虚拟机。

5.8 创建和管理路由的网络

路由模式不是很常见。在这种模式中，宿主机充当虚拟网络与外部物理网络之间的路由器，而且会自动配置正确的路由条目。这种模式的关键点是在宿主机之外的路由设置，需要用到虚拟网络的“返程”路由。

我们将根据图 5-32 所示的拓扑来做一个实验。在这个实验中，宿主机网卡 ens32 的 IP 地址是 192.168.1.231/24，需要创建的新虚拟网络的 IP 地址是 192.168.100.0/24。除了在宿主机上进行配置外，还需要在宿主机之外的设备上配置正确的路由条目，在此实验中，将在外部主机 192.168.1.48/24 上添加“将目标地址是 192.168.100.0/24 的数据包转发给 192.168.1.231”的路由条目。

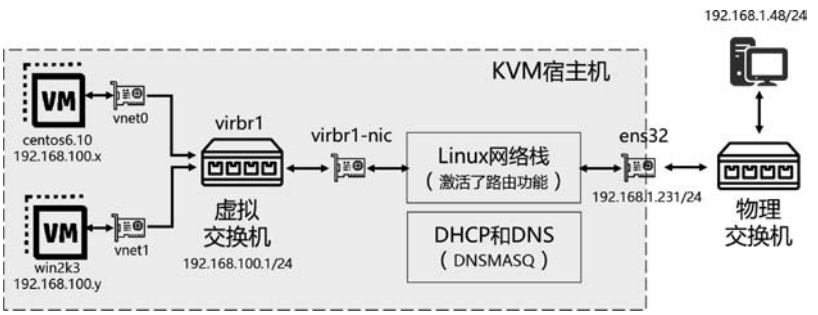

图 5-32 路由模式网络实验拓扑

5.8.1 在宿主机上创建路由模式的网络

当前版本的 Cockpit 虚拟化插件(cockpit-machines-211.3-1.el8.noarch)不支持创建路由模式的网络，但是可以正确地显示路由模式的网络。我们可以通过 virt-manager 来创建路由模式的网络。创建时，将 Mode 设置为 Routed，可以保持 Forward to 为 Any physical device，设置适合的 IPv4 configuration，如图 5-33 所示。

仍然使用 virsh 的 net-define 子命令来创建，创建时需要使用 XML 格式文件的内容如下：

```
<network>
  <name>route1</name>
  <forward mode="route"/>
```

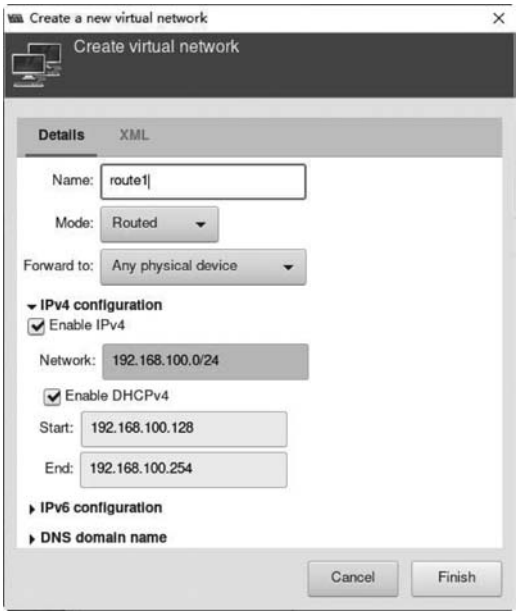

图 5-33　使用 virt-manager 创建路由模式网络

```
  <domain name = "route1"/>
  <ip address = "192.168.100.1" netmask = "255.255.255.0">
    <dhcp>
      <range start = "192.168.100.128" end = "192.168.100.254"/>
    </dhcp>
  </ip>
</network>
```

libvirt 在创建新的路由模式网络的时候，会在宿主机上创建新网桥及其子接口、增加路由中的路由条目、增加防火墙规则，执行的命令如下：

```
1 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
         valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
         valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff
      inet 192.168.1.231/24 brd 192.168.1.255 scope global noprefixroute ens32
         valid_lft forever preferred_lft forever
```

```
       inet6 fe80::20c:29ff:feb3:7336/64 scope link
          valid_lft forever preferred_lft forever
  3: virbr0: < NO - CARRIER, BROADCAST, MULTICAST, UP > mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
       link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
       inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
          valid_lft forever preferred_lft forever
  4: virbr0 - nic: < BROADCAST, MULTICAST > mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
       link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
  5: virbr1: < NO - CARRIER, BROADCAST, MULTICAST, UP > mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
       link/ether 52:54:00:d2:32:33 brd ff:ff:ff:ff:ff:ff
       inet 192.168.100.1/24 brd 192.168.100.255 scope global virbr1
          valid_lft forever preferred_lft forever
  6: virbr1 - nic: < BROADCAST, MULTICAST > mtu 1500 qdisc fq_codel master virbr1 state DOWN
group default qlen 1000
       link/ether 52:54:00:d2:32:33 brd ff:ff:ff:ff:ff:ff

2 # virsh net - list
   Name       State     Autostart     Persistent
  ----------------------------------------------------------
   default    active    yes           yes
   route1     active    yes           yes

3 # virsh net - info route1
  Name:       route1
  UUID:       e166c9be - b791 - 4ea4 - 814a - 6f1c99326eea
  Active:     yes
  Persistent: yes
  Autostart:  yes
  Bridge:     virbr1

4 # virsh net - dumpxml route1
  < network >
    < name > route1 </name >
    < uuid > e166c9be - b791 - 4ea4 - 814a - 6f1c99326eea </uuid >
    < forward mode = 'route'/>
    < bridge name = 'virbr1' stp = 'on' delay = '0'/>
    < mac address = '52:54:00:d2:32:33'/>
    < domain name = 'route1'/>
    < ip address = '192.168.100.1' netmask = '255.255.255.0'>
      < dhcp >
        < range start = '192.168.100.128' end = '192.168.100.254'/>
      </dhcp >
    </ip >
  </network >
```

第 1 行命令的输出内容显示了新网桥的名称 virbr1，其 IP 地址是 192.168.100.1/24。virbr1 有 1 个接口 virbr1-nic。

第 2 行命令的输出内容显示了当前虚拟网络的列表。

第 3 行、第 4 行命令的输出内容显示了新虚拟网络的详细信息。

接下来执行的命令如下：

```
5 #route -n
  Kernel IP routing table
  Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
  0.0.0.0         192.168.1.2     0.0.0.0         UG    100    0        0 ens32
  192.168.1.0     0.0.0.0         255.255.255.0   U     100    0        0 ens32
  192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr1
  192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

在第 5 行命令的输出中看到了新增加的路由条目：目标是 192.168.100.0/24 的数据将被发送给 virbr1，也就是新的虚拟网络。

接下来执行的命令如下：

```
6 #iptables -L -n -v
  Chain INPUT (policy ACCEPT 202 packets, 17430 Bytes)
    target     prot opt in     out     source               destination
    ACCEPT     udp  --  virbr1 *       0.0.0.0/0            0.0.0.0/0           udp dpt:53
    ACCEPT     tcp  --  virbr1 *       0.0.0.0/0            0.0.0.0/0           tcp dpt:53
    ACCEPT     udp  --  virbr1 *       0.0.0.0/0            0.0.0.0/0           udp dpt:67
    ACCEPT     tcp  --  virbr1 *       0.0.0.0/0            0.0.0.0/0           tcp dpt:67
    ACCEPT     udp  --  virbr0 *       0.0.0.0/0            0.0.0.0/0           udp dpt:53
    ACCEPT     tcp  --  virbr0 *       0.0.0.0/0            0.0.0.0/0           tcp dpt:53
    ACCEPT     udp  --  virbr0 *       0.0.0.0/0            0.0.0.0/0           udp dpt:67
    ACCEPT     tcp  --  virbr0 *       0.0.0.0/0            0.0.0.0/0           tcp dpt:67

  Chain FORWARD (policy ACCEPT 0 packets, 0 Bytes)
    target     prot opt in     out     source               destination
    ACCEPT     all  --  *      virbr1  0.0.0.0/0            192.168.100.0/24
    ACCEPT     all  --  virbr1 *       192.168.100.0/24     0.0.0.0/0
    ACCEPT     all  --  virbr1 virbr1  0.0.0.0/0            0.0.0.0/0
    REJECT     all  --  *      virbr1  0.0.0.0/0            0.0.0.0/0           reject-with
icmp-port-unreachable
    REJECT     all  --  virbr1 *       0.0.0.0/0            0.0.0.0/0           reject-with
icmp-port-unreachable
    ACCEPT     all  --  *      virbr0  0.0.0.0/0            192.168.122.0/24    ctstate
RELATED,ESTABLISHED
    ACCEPT     all  --  virbr0 *       192.168.122.0/24     0.0.0.0/0
    ACCEPT     all  --  virbr0 virbr0  0.0.0.0/0            0.0.0.0/0
```

```
    REJECT     all --  * virbr0             0.0.0.0/0     0.0.0.0/0       reject-with
icmp-port-unreachable
    REJECT     all -- virbr0 *              0.0.0.0/0     0.0.0.0/0       reject-with
icmp-port-unreachable
  Chain OUTPUT (policy ACCEPT 177 packets, 21663 Bytes)
    target prot opt in out source destination
    ACCEPT udp --  * virbr1 0.0.0.0/0 0.0.0.0/0 udp dpt:68
    ACCEPT udp --  * virbr0 0.0.0.0/0 0.0.0.0/0 udp dpt:68
```

从第 6 行命令的输出中可以看到新增加的防火墙规则。

(1) INPUT 链：允许 virbr1 入站的 DNS(53 端口)和 DHCP(67 端口)的包。

(2) FORWARD 链：允许源是 virbr1 出站及相关联的返回包，源与目标都是 virbr1 的包，拒绝 icmp-port-unreachable 类型的包。

(3) OUTPUT 链：允许 virbr1 出站的 DHCP(68 端口)的包。

5.8.2 使用路由模式的网络

将虚拟机 centos6.10 调整到新增加的虚拟网络，命令如下：

```
1 #virsh edit centos6.10
  …
      <interface type='network'>
        <mac address='52:54:00:32:f8:e2'/>
        <source network='route1'/>
        <model type='virtio'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
      </interface>
  …

2 #virsh start centos6.10

3 #virsh domiflist centos6.10
  Interface     Type      Source     Model     MAC
  -------------------------------------------------------
  vnet0         network   route1     virtio    52:54:00:32:f8:e2

4 #virsh domifaddr centos6.10
   Name      MAC address          Protocol     Address
  -------------------------------------------------------------------------
   vnet0     52:54:00:32:f8:e2    ipv4         192.168.100.182/24
```

第 1 行命令用于编辑虚拟机的配置文件，其核心属性是<source network='route1'/>。

第 2 行命令用于启动这台虚拟机。

第 3 行命令的输出显示此虚拟机已经切换到新的名为 route1 的虚拟网络中了。

第4行命令显示虚拟机的IP地址是192.168.100.182/24，它是由宿主机的DNSMASQ组件分配的。

接下来执行的命令如下：

```
5 # ssh 192.168.100.182 "route -n"
  root@192.168.100.182's password:
  Kernel IP routing table
  Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
  192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
  169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth0
  0.0.0.0         192.168.100.1   0.0.0.0         UG    0      0        0 eth0

6 # ssh 192.168.100.182 "ping -c 2 192.168.1.231"
  root@192.168.100.182's password:
  PING 192.168.1.231 (192.168.1.231) 56(84) Bytes of data.
  64 Bytes from 192.168.1.231: icmp_seq=1 ttl=64 time=0.300 ms
  64 Bytes from 192.168.1.231: icmp_seq=2 ttl=64 time=0.582 ms

  --- 192.168.1.231 ping statistics ---
  2 packets transmitted, 2 received, 0% packet loss, time 1002ms
  rtt min/avg/max/mdev = 0.300/0.441/0.582/0.141 ms

7 # ssh 192.168.100.182 "ping -c 2 192.168.1.48"
  root@192.168.100.182's password:
  PING 192.168.1.48 (192.168.1.48) 56(84) Bytes of data.

  --- 192.168.1.48 ping statistics ---
  2 packets transmitted, 0 received, 100% packet loss, time 11003ms
```

第5行命令的输出显示了虚拟机操作系统中的路由表，其缺省网关是192.168.100.1，也就是宿主中的virbr1。

第6行命令从虚拟机中ping宿主机的ens32的IP地址，由于宿主机知道192.168.100.0/24在哪里，所以可以正常返回响应。

第7行命令从虚拟机中ping宿主机之外的主机(192.168.1.48)，从输出信息中可以看到：没有收到响应的包(0 received)。

这是由于在外部主机192.168.1.48上没有到192.168.100.0/24的正确路由。

在本实验中，192.168.1.48是一台Windows 10的计算机，所以可以通过route print查看其路由表，命令如下：

```
C:\WINDOWS\system32>route print -4
===========================================================================
接口列表
  4...38 d5 47 b9 33 c6 ......Realtek PCIe GbE Family Controller
```

```
  1.........................Software Loopback Interface 1
===========================================================================

IPv4 路由表
===========================================================================
活动路由:
网络目标            网络掩码            网关            接口            跃点数
0.0.0.0             0.0.0.0             192.168.1.1     192.168.1.48     291
127.0.0.0           255.0.0.0           在链路上        127.0.0.1        331
127.0.0.1           255.255.255.255     在链路上        127.0.0.1        331
127.255.255.255     255.255.255.255     在链路上        127.0.0.1        331
192.168.1.0         255.255.255.0       在链路上        192.168.1.48     291
192.168.1.48        255.255.255.255     在链路上        192.168.1.48     291
192.168.1.255       255.255.255.255     在链路上        192.168.1.48     291
224.0.0.0           240.0.0.0           在链路上        127.0.0.1        331
224.0.0.0           240.0.0.0           在链路上        192.168.1.48     291
255.255.255.255     255.255.255.255     在链路上        127.0.0.1        331
255.255.255.255     255.255.255.255     在链路上        192.168.1.48 V 291
===========================================================================
永久路由:
网络地址            网络掩码            网关地址        跃点数
0.0.0.0             0.0.0.0             192.168.1.1     默认
===========================================================================
```

由于在路由表中没有与 192.168.100.0/24 相匹配的条目,所以它会将响应 192.168.100.0/24 的包发给默认网关(192.168.1.1),而不是宿主机的 IP 地址 192.168.1.231。在本实验中,可以通过添加静态路由来解决这个问题,命令如下:

```
C:\WINDOWS\system32>route add 192.168.100.0 MASK 255.255.255.0 192.168.1.231

C:\WINDOWS\system32>route print -4 | find "192.168.100"
    192.168.100.0    255.255.255.0    192.168.1.231    192.168.1.48    36
```

这样就可以 ping 通了,命令如下:

```
9 # ssh 192.168.100.182 "ping -c 2 192.168.1.48"
  root@192.168.100.182's password:
  PING 192.168.1.48 (192.168.1.48) 56(84) Bytes of data.
  64 Bytes from 192.168.1.48: icmp_seq=1 ttl=127 time=0.857 ms
  64 Bytes from 192.168.1.48: icmp_seq=2 ttl=127 time=1.86 ms

  --- 192.168.1.48 ping statistics ---
  2 packets transmitted, 2 received, 0% packet loss, time 1004ms
  rtt min/avg/max/mdev = 0.857/1.363/1.869/0.506 ms
```

5.9 创建和管理开放的网络

开放模式与路由模式类似，其主要区别是 libvirt 不会为开放模式设置防火墙规则。下面通过实验来查看这种模式，命令如下：

```
1 #vi new.xml
  <network>
    <name>open1</name>
    <forward mode="open"/>
    <domain name="open1"/>
    <ip address="192.168.100.1" netmask="255.255.255.0">
      <dhcp>
        <range start="192.168.100.128" end="192.168.100.254"/>
      </dhcp>
    </ip>
  </network>

2 #virsh net-define new.xml

3 #virsh net-autostart open1

4 #virsh net-start open1

5 #virsh net-list
   Name       State     Autostart     Persistent
  ----------------------------------------------------------
   default    active    yes           yes
   open1      active    yes           yes
```

第 1 行命令用于创建新网络的 XML 定义文件。

第 2 行命令通过 XML 文件定义新网络，第 3 行命令用于设置自动启动，第 4 行命令用于启动这个开放式网络。

第 5 行命令的输出显示新网络创建成功了。

接下来执行的命令如下：

```
6 #virsh net-info open1
  Name:        open1
  UUID:        55dfc4da-dccf-47b7-a2b9-d2f5c130804c
  Active:      yes
  Persistent:  yes
  Autostart:   yes
  Bridge:      virbr1
```

```
7 # virsh net-dumpxml open1
  <network>
    <name>open1</name>
    <uuid>55dfc4da-dccf-47b7-a2b9-d2f5c130804c</uuid>
    <forward mode='open'/>
    <bridge name='virbr1' stp='on' delay='0'/>
    <mac address='52:54:00:16:a8:57'/>
    <domain name='open1'/>
    <ip address='192.168.100.1' netmask='255.255.255.0'>
      <dhcp>
        <range start='192.168.100.128' end='192.168.100.254'/>
      </dhcp>
    </ip>
  </network>
```

通过第 6 行、第 7 行命令可以查看新网络的信息。

接下来执行的命令如下：

```
8 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
         valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
         valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:f7:6b:c8 brd ff:ff:ff:ff:ff:ff
      inet 192.168.1.48/24 brd 192.168.1.255 scope global noprefixroute ens32
         valid_lft forever preferred_lft forever
      inet6 fe80::20c:29ff:fef7:6bc8/64 scope link
         valid_lft forever preferred_lft forever
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
      inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
         valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
      link/ether 52:54:00:e0:41:ac brd ff:ff:ff:ff:ff:ff
  5: virbr1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:16:a8:57 brd ff:ff:ff:ff:ff:ff
      inet 192.168.100.1/24 brd 192.168.100.255 scope global virbr1
         valid_lft forever preferred_lft forever
```

```
    6: virbr1-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr1 state DOWN
group default qlen 1000
    link/ether 52:54:00:16:a8:57 brd ff:ff:ff:ff:ff:ff
```

从第 8 行命令的输出可以看出：新增加了 1 个名为 virbr1 的网桥，其 IP 地址是 192.168.100.1/24，它有一个名为 virbr1-nic 的子接口。

接下来执行的命令如下：

```
9 #iptables -L -n -v
  Chain INPUT (policy ACCEPT 963 packets, 70270 Bytes)
   target     prot opt in     out source     destination
   ACCEPT     udp -- virbr0 *     0.0.0.0/0     0.0.0.0/0     udp dpt:53
   ACCEPT     tcp -- virbr0 *     0.0.0.0/0     0.0.0.0/0     tcp dpt:53
   ACCEPT     udp -- virbr0 *     0.0.0.0/0     0.0.0.0/0     udp dpt:67
   ACCEPT     tcp -- virbr0 *     0.0.0.0/0     0.0.0.0/0     tcp dpt:67

  Chain FORWARD (policy ACCEPT 0 packets, 0 Bytes)
   target prot opt in     out     source          destination
   ACCEPT all     --  * virbr0     0.0.0.0/0     192.168.122.0/24     ctstate
RELATED,ESTABLISHED
   ACCEPT all     -- virbr0 *     192.168.122.0/24     0.0.0.0/0
   ACCEPT all     -- virbr0 virbr0     0.0.0.0/0     0.0.0.0/0
   REJECT all     --  * virbr0     0.0.0.0/0     0.0.0.0/0     reject-with
icmp-port-unreachable
   REJECT all     -- virbr0 *     0.0.0.0/0     0.0.0.0/0     reject-with
icmp-port-unreachable

  Chain OUTPUT (policy ACCEPT 508 packets, 62659 Bytes)
   target prot opt in     out     source     destination
   ACCEPT udp --  *     virbr0     0.0.0.0/0     0.0.0.0/0     udp dpt:68
```

与路由模式不同，libvirt 不会为开放模式的网络设置防火墙规则，所以不会在第 9 行命令的输出中看到与 192.168.100.0/24 和 virbr1 相关的策略。

接下来执行的命令如下：

```
10 #virsh edit centos6.10
  …
  <interface type='network'>
    <mac address='52:54:00:27:5f:c9'/>
    <source network='open1'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
…
```

```
11 #virsh start centos6.10
```

第 10 行命令用于修改虚拟机 centos6.10 的网卡属性，使其连接到 open1 这个开放的网络。

第 11 行命令用于启动虚拟机。

默认情况下，即使为 open1 网络配置 DHCP 地址范围（192.168.100.128 至 192.168.100.254），虚拟机 centos6.10 也无法得到 IP 地址，如图 5-34 所示。

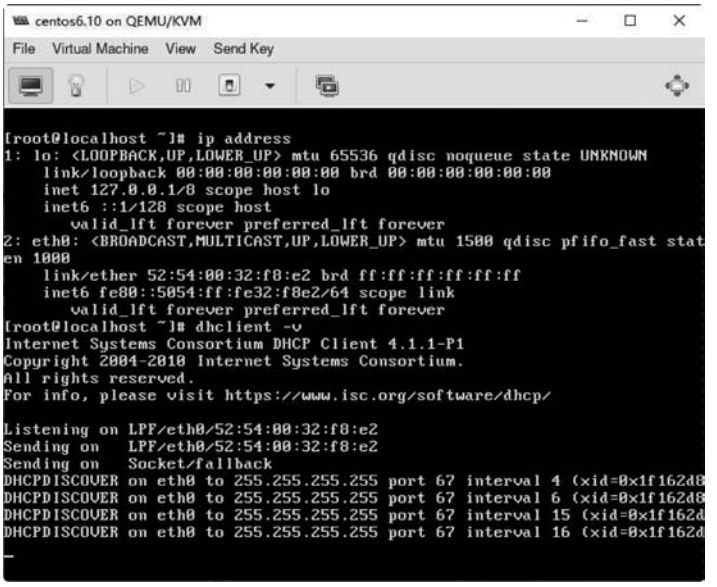

图 5-34　开放模式网络中虚拟机无法通过 DHCP 获得 IP 地址

接下来执行的命令如下：

```
12 #virsh net-dhcp-leases open1
 Expiry Time     MAC address     Protocol     IP address  Hostname  Client ID or DUID
-------------------------------------------------------------------------------
```

第 12 行命令的输出也显示了这个名为 open1 的网络并没有为虚拟机分配 IP 地址。

出现这种现象的原因是：libvirt 并没有为 192.168.100.0/24 和 virbr1 设置允许 DHCP 数据的规则，所以虚拟机发出的 DHCPDISCOVER 包并没有被宿主机上的 DNSMASQ 组件收到。

如果希望为 open1 中的虚拟机自动分配 IP 地址，就需要手工添加防火墙规则。命令如下：

```
13 #firewall-cmd --list-all
   public (active)
     target: default
     icmp-block-inversion: no
     interfaces: ens32
     sources:
     services: cockpit dhcpv6-client ssh
     ports:
     protocols:
     masquerade: no
     forward-ports:
     source-ports:
     icmp-blocks:
     rich rules:

14 #firewall-cmd --add-interface=virbr1

15 #firewall-cmd --add-service=dhcp

16 #firewall-cmd --list-all
   public (active)
     target: default
     icmp-block-inversion: no
     interfaces: ens32 virbr1
     sources:
     services: cockpit dhcp dhcpv6-client ssh
     ports:
     protocols:
     masquerade: no
     forward-ports:
     source-ports:
     icmp-blocks:
     rich rules:
```

RHEL/CentOS 8 推荐使用 firewall 的 firewall-cmd 命令来管理防火墙。通过第 13 行命令来获得当前防火墙配置的概要信息。

第 14 行命令向 public 区域添加了一个接口 virbr1。第 15 行命令又允许 DHCP 服务，这样将允许来自 open1 网络的与 DHCP 相关的数据包到达宿主机，DNSMASQ 收到后就会为其分配 IP 地址。

接下来执行的命令如下：

```
17 #virsh net-dhcp-leases open1
   Expiry Time     MAC address     Protocol IP address     Hostname
Client ID or DUID
   -------------------------------------------------------------------
   2020-12-01  20:56:03     52:54:00:32:f8:e2  ipv4     192.168.100.182/24
```

第 17 行命令的输出显示：向虚拟机分配的 IP 地址是 192.168.100.182/24。

后续的配置与路由模式类似，这里就不再赘述了。

提示：如果需要使防火墙配置永久生效，还需要使用--permanent 选项来执行 firewall-cmd 命令。

在配置路由模式时，libvirt 会为其配置防火墙规则，例如允许 DHCP、DNS 等流量，而在配置开放模式时，libvirt 不会为此虚拟网络生成任何 iptables 规则，这样就需要用户自行配置了。

5.10 实现多 VLAN 支持

虚拟局域网(Virtual LAN，VLAN)是物理网络中的逻辑网络，它提高了网络管理的安全性与灵活性。通过使用 VLAN 标记，可以将每个以太网端口都视为包含许多逻辑端口。通过合理的规划与配置，可以在一台 KVM 宿主机中运行属于不同 VLAN 的虚拟机，如图 5-35 所示。

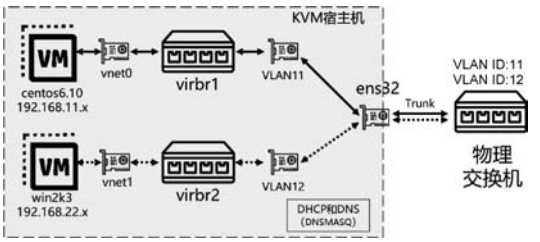

图 5-35 同一个宿主机中可以运行属于多个不同 VLAN 的虚拟机

图 5-35 中的物理网络有两个 VLAN，其 ID 分别是 11 和 12。虚拟机 centos6.10 属于 ID 为 11 的 VLAN，虚拟机 Win2k3 则属于 ID 为 12 的 VLAN。

首先，需要在宿主机现有接口(例如以太网卡、绑定网卡或网桥设备)上创建 VLAN 接口，在图 5-35 中这个接口是物理网卡 ens32。在 ens32 上创建两个使用不同 VLAN ID 的虚拟网络接口 VLAN11 和 VLAN12，ens32 被称为 VLAN11、VLAN12 的“父”接口。

然后创建两个网桥 virbr1 和 virbr2。其中 virbr1 的子接口是 VLAN11，virbr2 的子接口是 VLAN12。

将虚拟机 centos6.10 桥接到 virbr1，而虚拟机 Win2k3 则需要桥接到 virbr2。

在宿主机所在的物理网络中也需要进行设置，主要就是将宿主机网卡 ens32 所连接的交换机端口配置为 trunk 模式。

5.10.1 创建支持 VLAN 的网络接口

Linux 通过在内核中加载 8021q 模块实现 VLAN 功能。在 RHEL/CentOS 8 中，默认

情况下会根据需要自动加载 8021q 模块。在其他的 Linux 发版本中，可能需要使用 modprobe 命令来手工加载。

在 RHEL/CentOS 8 中，既可以通过 iproute2、NetworkManager、Cockpit 等多种工具来管理 VLAN，也可以直接编辑/etc/sysconfig/network-scripts/目录下的网络接口脚本文件实现 VLAN 的管理。由于 iproute2 中的 ip 命令所创建的 VLAN 不是永久性的，所以不推荐使用。下面，我们通过 nmcli 和 Cockpit 这两种工具来创建和管理 VLAN，命令如下：

```
1 # ip address
  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
         valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
         valid_lft forever preferred_lft forever
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff
      inet 192.168.1.231/24 brd 192.168.1.255 scope global noprefixroute ens32
         valid_lft forever preferred_lft forever
      inet6 fe80::20c:29ff:feb3:7336/64 scope link
         valid_lft forever preferred_lft forever
  3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
default qlen 1000
      link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff
      inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
         valid_lft forever preferred_lft forever
  4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN
group default qlen 1000
      link/ether 52:54:00:fd:b2:60 brd ff:ff:ff:ff:ff:ff

2 # ls /etc/sysconfig/network-scripts/
  ifcfg-ens32

3 # cat /etc/sysconfig/network-scripts/ifcfg-ens32
  TYPE=Ethernet
  NAME=ens32
  DEVICE=ens32
  ONBOOT=yes
  IPADDR=192.168.1.231
  PREFIX=24
  GATEWAY=192.168.1.1
  DNS1=8.8.8.8
```

通过第 1 行命令查看当前网络接口的地址信息。

第 2 行命令显示在/etc/sysconfig/network-scripts/目录中仅包含 ens32 的接口配置文件 ifcfg-ens32。通过第 3 行命令查看文件的内容，这是 1 个精简后的配置文件，仅保留必要的设置值。

接下来执行的命令如下：

```
4 # nmcli connection add type vlan con - name vlan11 ifname vlan11 vlan.parent ens32 vlan.id 11
  Connection 'vlan11' (76131bf6 - 2889 - 4e44 - 9a4a - 90281bd2e22a) successfully added.

5 # nmcli connection modify vlan11 ipv4.addresses '192.168.11.231/24' ipv4.gateway 192.168.
11.1 ipv4.method manual

6 # nmcli connection up vlan11
    Connection successfully activated (D - Bus active path: /org/freedesktop/NetworkManager/
ActiveConnection/10)
```

我们先通过 NetworkManager 的命令行工具 nmcli 来创建新连接。

第 4 行命令创建了一个新的连接，通过 type 指定类型为 vlan，通过 con-name 指定新连接的名称为 vlan11，通过 ifname 指定的接口名称也是 vlan11，vlan. parent 指定父接口是 ens32，vlan. id 指定 VLAN 的 ID 是 11。

第 5 行命令用于设置 VLAN 网络接口的 IPv4 地址、网关和分配地址的模式。

第 6 行命令激活这个连接。

接下来执行的命令如下：

```
7 # ip - d address show vlan11
  10: vlan11@ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
      link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 0 maxmtu 65535
      vlan protocol 802.1Q id 11 <REORDER_HDR> numtxqueues 1 numrxqueues 1 gso_max_size 65536
gso_max_segs 65535
      inet 192.168.11.231/24 brd 192.168.11.255 scope global noprefixroute vlan11
         valid_lft forever preferred_lft forever
      inet6 fe80::1db2:9a6c:6979:894e/64 scope link noprefixroute
         valid_lft forever preferred_lft forever

8 # ip - d address show ens32
  2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
      link/ether 00:0c:29:b3:73:36 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 46 maxmtu
16110 numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
      inet 192.168.1.231/24 brd 192.168.1.255 scope global noprefixroute ens32
         valid_lft forever preferred_lft forever
      inet6 fe80::20c:29ff:feb3:7336/64 scope link
         valid_lft forever preferred_lft forever
```

在第7行命令中使用了-d选项，会显示连接的详细信息。自动生成的连接名称“vlan11@ens32”很直观，表示vlan11是在ens32之上实现的。除了IP地址等基本信息之外，“vlan protocol 802.1Q id 11”显示了VLAN的协议和ID号。

对比一下第8行命令的输出，会发现新创建的vlan11“借用”了ens32的MAC地址“00:0c:29:b3:73:36”。

接下来执行的命令如下：

```
9 #cat /etc/sysconfig/network-scripts/ifcfg-vlan11
  VLAN=yes
  TYPE=Vlan
  PHYSDEV=ens32
  VLAN_ID=11
  REORDER_HDR=yes
  GVRP=no
  MVRP=no
  HWADDR=
  PROXY_METHOD=none
  BROWSER_ONLY=no
  BOOTPROTO=none
  DEFROUTE=yes
  IPV4_FAILURE_FATAL=no
  IPV6INIT=yes
  IPV6_AUTOCONF=yes
  IPV6_DEFROUTE=yes
  IPV6_FAILURE_FATAL=no
  IPV6_ADDR_GEN_MODE=stable-privacy
  NAME=vlan11
  UUID=76131bf6-2889-4e44-9a4a-90281bd2e22a
  DEVICE=vlan11
  ONBOOT=yes
  IPADDR=192.168.11.231
  PREFIX=24
  GATEWAY=192.168.11.1
```

NetworkManager创建新的VLAN接口，还会在/etc/sysconfig/network-scripts/目录中创建网络接口配置文件，第9行命令显示了这个配置文件的内容，其中最重要的属性有以下几种。

（1）VLAN=yes。

（2）TYPE=Vlan。

（3）PHYSDEV=ens32。

（4）VLAN_ID=11。

（5）DEVICE=vlan11。

（6）ONBOOT＝yes。

提示：可以通过 man nm-settings-ifcfg-rh 来获得/etc/sysconfig/network-scripts/ifcfg-* 设置的详细说明。

接下来执行的命令如下：

```
10 #lsmod | grep 8021q
   8021q    40960    0
   garp     16384    1  8021q
   mrp      20480    1  8021q
```

当激活 VLAN 网络接口时，NetworkManager 会自动将 8021q 模块加载到内核中，可以从第 10 行命令的输出中看到这个模块。

相对于 nmcli，通过 Cockpit 来创建新的 VLAN 接口特别简单。

（1）在 Cockpit 左边导航中单击 Networking，会显示当前的所有网络接口，如图 5-36 所示。

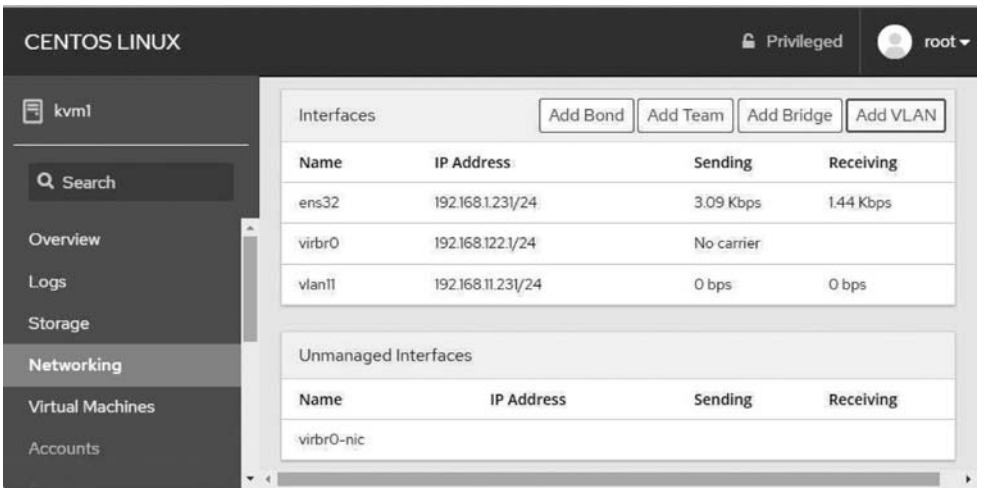

图 5-36　Cockpit 中的网络接口信息

（2）单击 Add VLAN 按钮。

（3）在 VLAN Setting 对话框中，选择要为其创建 VLAN 的物理接口 ens32，将 VLAN ID 设置为 12，将接口名称设置为 vlan12，然后单击 Apply 按钮完成创建，如图 5-37 所示。

（4）新 VLAN 创建完毕，如图 5-38 所示。单击 Configure IP 链接配置网络设置。

（5）在 IPv4 Setting 对话框中设置 IP 地址及缺省网关等信息，然后单击 Apply 按钮保存配置，如图 5-39 所示。

图 5-37　设置新 VLAN 网络接口的信息

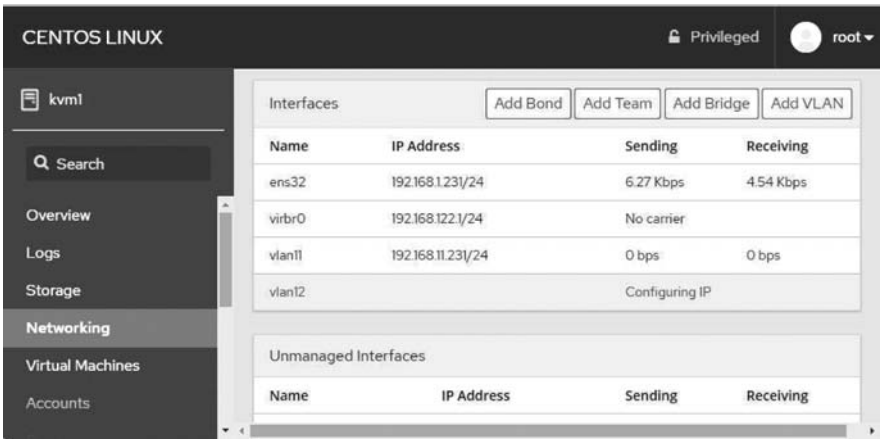

图 5-38　包括新网络接口的列表

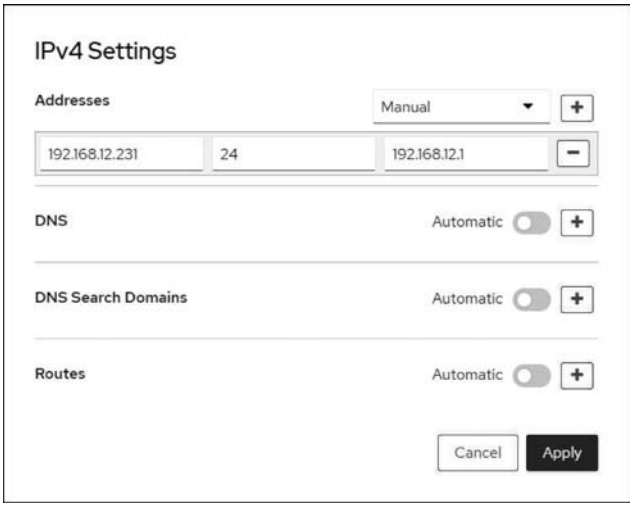

图 5-39　为新的网络接口设置 IP 地址等信息

Cockpit 还会在/etc/sysconfig/network-scripts/目录中创建网络接口配置文件 ifcfg-vlan12，命令如下：

```
11 #cat /etc/sysconfig/network-scripts/ifcfg-vlan12
   VLAN=yes
   TYPE=Vlan
   PHYSDEV=ens32
   VLAN_ID=12
   REORDER_HDR=yes
   GVRP=no
   MVRP=no
   HWADDR=
   PROXY_METHOD=none
   BROWSER_ONLY=no
   BOOTPROTO=none
   DEFROUTE=yes
   IPV4_FAILURE_FATAL=no
   IPV6INIT=yes
   IPV6_AUTOCONF=yes
   IPV6_DEFROUTE=yes
   IPV6_FAILURE_FATAL=no
   IPV6_ADDR_GEN_MODE=stable-privacy
   NAME=vlan12
   UUID=ae59cf5b-f088-4ba6-9f20-7eb2def383fe
   DEVICE=vlan12
   ONBOOT=yes
   IPADDR=192.168.12.231
   PREFIX=24
   GATEWAY=192.168.12.1
   PEERDNS=no
   PEERROUTES=no
```

提示：也可以通过在/etc/sysconfig/network-scripts/目录中创建新的配置文件的方法来创建新的支持的 VLAN 接口。

5.10.2 创建使用 VLAN 网络接口的网桥

有了两个支持 VLAN 的网络接口 VLAN11 和 VLAN12，根据图 5-35 所示的拓扑，我们需要在宿主机上创建两个网桥 virbr1 和 virbr2。VLAN11 是 virbr1 的子接口，VLAN12 是 virbr2 的子接口。

与前面创建网桥的方法类似，最简单的方法是通过 Cockpit 来创建。

(1) 在 Cockpit 左边导航中单击 Networking，会显示当前的所有网络接口，如图 5-40 所示。

（2）单击 Add Bridge 按钮。

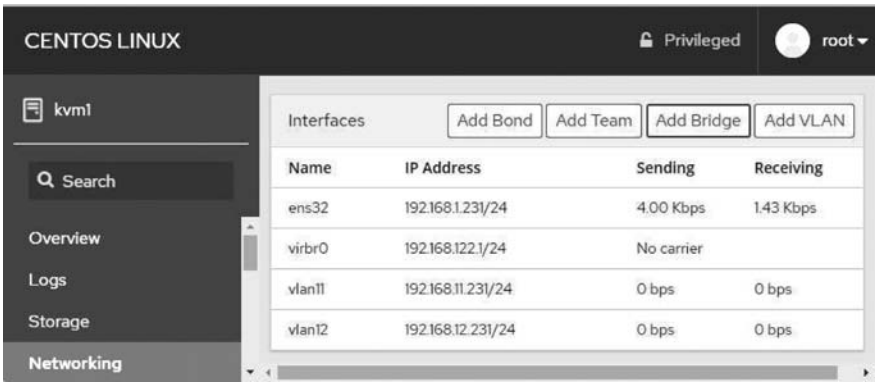

图 5-40　Cockpit 中的网络接口信息

（3）在 Bridge Setting 对话框中，将网桥名设置为 virbr1，选择子接口 vlan11，然后单击 Apply 按钮完成创建，如图 5-41 所示。

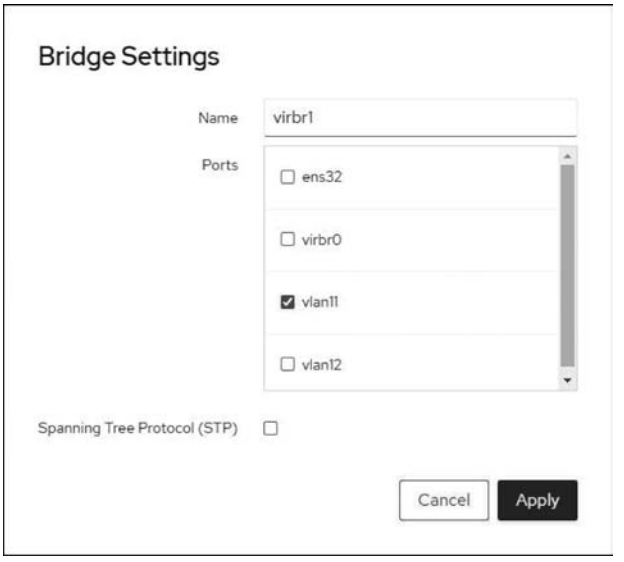

图 5-41　设置新网桥的信息

（4）类似地，再创建使用 VLAN12 的新网桥 virbr2，最后的结果如图 5-42 所示。

5.10.3　配置虚拟机使用 VLAN

将虚拟机 centos6.10 桥接到 virbr1，而虚拟机 win2k3 则需要桥接到 virbr2。如果使用的是 Cockpit，则可在 Interface Type 选择 Bridge to LAN，在 Source 中选择适当的网桥，如图 5-43 所示。

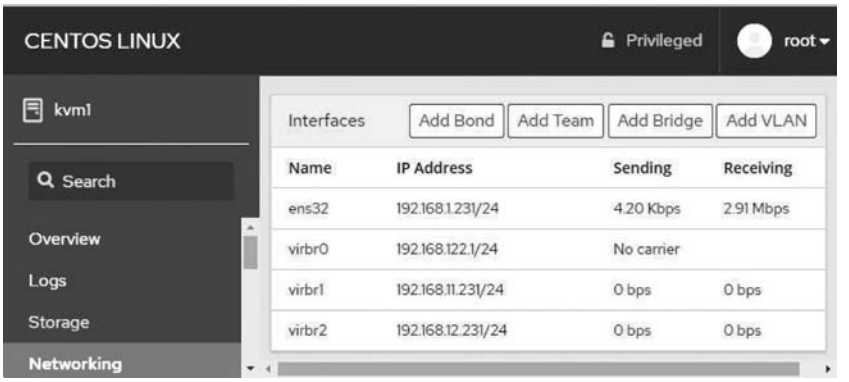

图 5-42　包括新网桥的列表

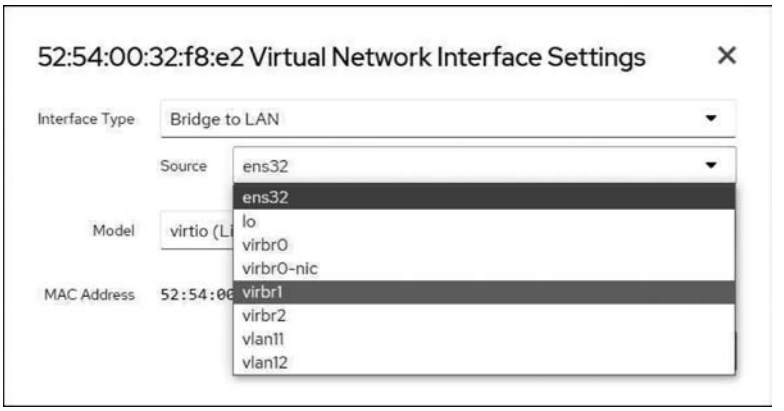

图 5-43　使用 Cockpit 更新虚拟机的网络配置

也可以通过 virsh 的 edit 等子命令来修改虚拟机的配置文件，最重要的是< source bridge='virbr1'/>这一行的设置，配置文件的内容如下：

```
    <interface type='bridge'>
      <mac address='52:54:00:32:f8:e2'/>
      <source bridge='virbr1'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </interface>
```

在此实验环境中，我们可以使用 Wireshark 等协议分析工具在宿主机外面的网络中抓到来自虚拟机而且打有 VLAN 标记的数据包，如图 5-44 所示。

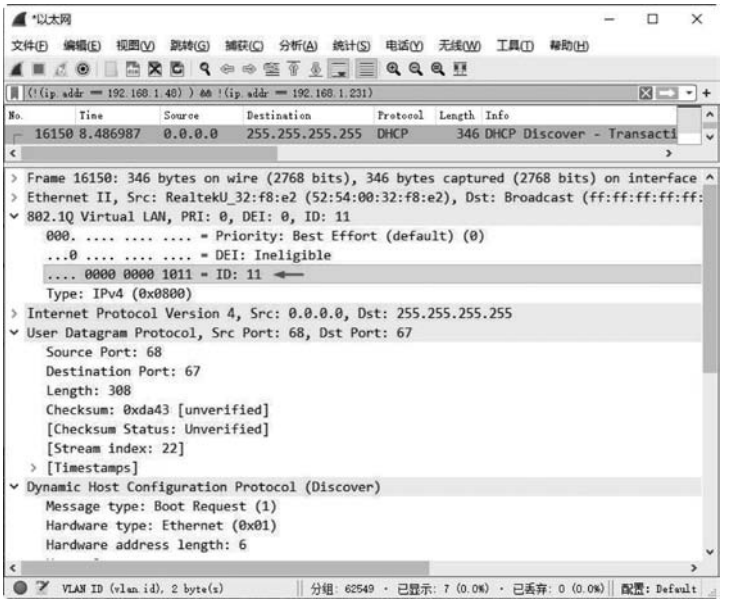

图 5-44　来自虚拟机 centos6.10 并带有 VLAN 标记的数据包

5.11　通过网络过滤器提高安全性

网络过滤器(Network Filter)是 libvirt 的一个子系统,可以通过在其中配置过滤规则来对每个虚拟机网络接口的流量进行控制,从而提高安全性。libvirt 会将这些过滤规则转换为宿主机上的防火墙规则,在虚拟机启动时自动添加与其网络接口有关的防火墙规则,而当其关闭时也会自动删除这些规则。

与虚拟机内部的防火墙不同,网络过滤规则应用于宿主机之上,所以无法从虚拟机内部规避这些过滤规则,因此从虚拟机用户的角度来看,这些规则是透明的、强制性的、无法“旁路”的,如图 5-45 所示。

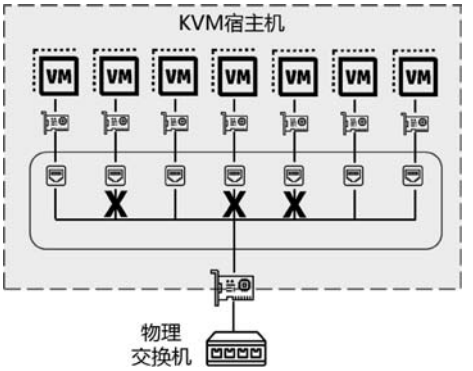

图 5-45　网络过滤器原理

既可以针对特定虚拟机的某个网络接口配置网络过滤规则，也可以让多个虚拟机使用相同的过滤规则。

5.11.1 网络过滤器基本原理

下面通过一个简单的实验来了解网络过滤器的基本原理。

在实验中，创建一个阻止所有 ICMP 流量的过滤器，然后将其应用于虚拟机 centos6.10。

首先，查看虚拟机 centos6.10 的网络连通性，命令如下：

```
1 # virsh domifaddr centos6.10
   Name     MAC address          Protocol     Address
  -------------------------------------------------------------------------------
   vnet0    52:54:00:27:5f:c9  ipv4         192.168.122.142/24

2 # ping -c 2 192.168.122.142
  PING 192.168.122.142 (192.168.122.142) 56(84) Bytes of data.
  64 Bytes from 192.168.122.142: icmp_seq=1 ttl=64 time=0.939 ms
  64 Bytes from 192.168.122.142: icmp_seq=2 ttl=64 time=1.20 ms

  --- 192.168.122.142 ping statistics ---
  2 packets transmitted, 2 received, 0% packet loss, time 3ms
  rtt min/avg/max/mdev = 0.939/1.071/1.204/0.136 ms

3 # iptables -L -n
  Chain INPUT (policy ACCEPT)
  target     prot opt source     destination
  ACCEPT     udp -- 0.0.0.0/0   0.0.0.0/0     udp dpt:53
  ACCEPT     tcp -- 0.0.0.0/0   0.0.0.0/0     tcp dpt:53
  ACCEPT     udp -- 0.0.0.0/0   0.0.0.0/0     udp dpt:67
  ACCEPT     tcp -- 0.0.0.0/0   0.0.0.0/0     tcp dpt:67

  Chain FORWARD (policy ACCEPT)
  target     prot opt source     destination
  ACCEPT     all -- 0.0.0.0/0   192.168.122.0/24   ctstate RELATED,ESTABLISHED
  ACCEPT     all -- 192.168.122.0/24   0.0.0.0/0
  ACCEPT     all -- 0.0.0.0/0   0.0.0.0/0
  REJECT     all -- 0.0.0.0/0   0.0.0.0/0   reject-with icmp-port-unreachable
  REJECT     all -- 0.0.0.0/0   0.0.0.0/0   reject-with icmp-port-unreachable

  Chain OUTPUT (policy ACCEPT)
  target     prot opt source     destination
  ACCEPT     udp -- 0.0.0.0/0   0.0.0.0/0     udp dpt:68
```

通过第 1 行命令的输出可以看出：虚拟机 centos6.10 网络接口在宿主机上的名称为 vnet0，IP 地址为 192.168.122.142。

第 2 行命令的输出显示了可以在宿主机上 ping 通虚拟机，这说明虚拟机操作系统的防火墙没有阻止 ICMP 的 echo 请求与响应。

第 3 行命令的输出显示了宿主机上当前防火墙的配置，其中并没有针对 vnet0 或 192.168.122.142 的特别设置。

接下来执行的命令如下：

```
4 # vi new.xml
  <filter name = 'blockicmp'>
    <rule action = 'drop' direction = 'in'>
      <icmp/>
    </rule>
  </filter>

5 # virsh nwfilter - define new.xml
  Network filter blockicmp defined from new.xml

6 # virsh nwfilter - list
   UUID                                          Name
   ----------------------------------------------------------------------------
   4b69faf1 - fac2 - 4efe - 9f30 - d8bd12d0de87   allow - arp
   bdaaee09 - 70ff - 4bff - a0c5 - f1d9313c19df   allow - dhcp
   93a8a4ca - 2714 - 4d93 - 8df3 - 1610bf0eac06   allow - dhcp - server
   fe8d7784 - 7b2d - 4d8e - 9aa4 - 642ca8e282f4   allow - incoming - ipv4
   34ab7687 - 59b0 - 414e - 9294 - 4884dfe7c111   allow - ipv4
   737777d6 - 1115 - 425a - 9063 - defc0e14b614   blockicmp
   da713eaa - e490 - 4d61 - 8489 - b62110795ce4   clean - traffic
   489f315a - 4ed5 - 4fa3 - 8336 - 9acebc1dee83   clean - traffic - gateway
   843b6276 - 68a2 - 4bd8 - 8a31 - 76efe4ee7526   no - arp - ip - spoofing
   0c6a9780 - f1e5 - 4181 - a758 - c1aa58321ff0   no - arp - mac - spoofing
   e9dd8d7d - 0fde - 4cb5 - 9917 - b192d761782b   no - arp - spoofing
   76da0b46 - fc7c - 4991 - b421 - f18210a4887a   no - ip - multicast
   724ec182 - c6fe - 4289 - a1d3 - 8b0b4821a897   no - ip - spoofing
   d8a940e2 - ebdf - 4ffd - 9c85 - 75023362829f   no - mac - broadcast
   5036c658 - 6bea - 4299 - aff4 - 0bd53a9048dd   no - mac - spoofing
   a4b3fc68 - 629e - 45a3 - b42a - 73c6b1b19879   no - other - l2 - traffic
   397ccccf - c425 - 4c93 - 98f9 - 35fb3870c61c   no - other - rarp - traffic
   746cf6aa - ff87 - 497d - a0f6 - 89352bde27b0   qemu - announce - self
   a3b67cb4 - e977 - 46db - a084 - 24e9b9e74b63   qemu - announce - self - rarp
```

为了创建网络过滤器，需要创建一个 XML 格式的文件，其内容如第 4 行命令所示。< filter name＝'blockicmp'>指定新的过滤器的名称为 blockicmp，并且通过< rule action＝'drop' direction＝'in'>来创建一个规则，action＝'drop'表示这是一个阻止规则，direction＝'in'表示只针对入站的流量，< icmp/>用于指定协议为 icmp。

第 5 行命令使用 virsh 的 nwfilter-define 子命令来创建网络过滤器。

第6行命令的输出显示了当前系统中网络过滤器的列表。除了新的网络过滤器blockicmp之外，还会看到多个预安装的过滤器。

提示：virsh有多个与网络过滤器有关的并以nwfilter-开头的子命令，nwfilter是Network Filter的缩写。

接下来执行的命令如下：

```
7 #virsh shutdown centos6.10

8 #virsh edit centos6.10
      …
      <interface type='network'>
        <mac address='52:54:00:27:5f:c9'/>
        <source network='default'/>
        <model type='virtio'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
        <filterref filter='blockicmp'/>
      </interface>
      …

9 #virsh start centos6.10
```

第7行命令用于关闭虚拟机。

第8行命令用于修改虚拟机的配置文件。通过<filterref filter='blockicmp'/>将虚拟机的网络接口与网络过滤器blockicmp相关联。

第9行命令用于启动虚拟机。

接下来执行的命令如下：

```
10 #ping -c 2 192.168.122.142
   PING 192.168.122.142 (192.168.122.142) 56(84) Bytes of data.

   --- 192.168.122.142 ping statistics ---
   2 packets transmitted, 0 received, 100% packet loss, time 27ms

11 #ssh 192.168.122.142
   root@192.168.122.142's password:
   Last login: Thu Dec 17 11:29:33 2020 from 192.168.122.1
   [root@localhost ~]#exit
   logout
   Connection to 192.168.122.142 closed.
```

虚拟机启动之后，可在宿主机上测试其与虚拟机的连通性。从第10行命令的输出可以看

出：无法在宿主机上 ping 通虚拟机，这是由于网络过滤器 blockicmp 阻止了所有的 ICMP 流量。

不过，由于网络过滤器 blockicmp 仅仅阻止了 ICMP 流量，所以可以通过 SSH 访问这台虚拟机，如第 11 行命令的输出所示。

接下来执行的命令如下：

```
12 # iptables -L -n
   Chain INPUT (policy ACCEPT)
   target              prot opt source            destination
   libvirt-host-in     all -- 0.0.0.0/0           0.0.0.0/0
   ACCEPT              udp -- 0.0.0.0/0           0.0.0.0/0      udp dpt:53
   ACCEPT              tcp -- 0.0.0.0/0           0.0.0.0/0      tcp dpt:53
   ACCEPT              udp -- 0.0.0.0/0           0.0.0.0/0      udp dpt:67
   ACCEPT              tcp -- 0.0.0.0/0           0.0.0.0/0      tcp dpt:67

   Chain FORWARD (policy ACCEPT)
   target              prot opt source            destination
   libvirt-in          all -- 0.0.0.0/0           0.0.0.0/0
   libvirt-out         all -- 0.0.0.0/0           0.0.0.0/0
   libvirt-in-post     all -- 0.0.0.0/0           0.0.0.0/0
   ACCEPT              all -- 0.0.0.0/0           192.168.122.0/24    ctstate
RELATED,ESTABLISHED
   ACCEPT              all -- 192.168.122.0/24    0.0.0.0/0
   ACCEPT              all -- 0.0.0.0/0           0.0.0.0/0
   REJECT              all -- 0.0.0.0/0           0.0.0.0/0      reject-with
icmp-port-unreachable
   REJECT              all -- 0.0.0.0/0           0.0.0.0/0      reject-with
icmp-port-unreachable

   Chain OUTPUT (policy ACCEPT)
   target     prot opt source              destination
   ACCEPT     udp -- 0.0.0.0/0             0.0.0.0/0     udp dpt:68

   Chain libvirt-in (1 references)
   target     prot opt source     destination
   FI-vnet0 all -- 0.0.0.0/0 0.0.0.0/0     [goto] PHYSDEV match -- physdev-in vnet0

   Chain libvirt-out (1 references)
   target     prot opt source     destination
   FO-vnet0 all -- 0.0.0.0/0 0.0.0.0/0     [goto] PHYSDEV match -- physdev-out vnet0 --
physdev-is-bridged

   Chain libvirt-in-post (1 references)
   target     prot opt source                 destination
   ACCEPT     all -- 0.0.0.0/0 0.0.0.0/0     PHYSDEV match -- physdev-in vnet0
```

```
Chain libvirt-host-in (1 references)
target     prot opt source          destination
HI-vnet0 all  --  0.0.0.0/0       0.0.0.0/0     [goto] PHYSDEV match --physdev-in vnet0

Chain FO-vnet0 (1 references)
target     prot opt source        destination
DROP       icmp --  0.0.0.0/0     0.0.0.0/0

Chain FI-vnet0 (1 references)
target      prot opt source        destination
DROP        icmp --  0.0.0.0/0     0.0.0.0/0

Chain HI-vnet0 (1 references)
target       prot opt source        destination
DROP         icmp --  0.0.0.0/0     0.0.0.0/0
```

第12行命令的输出显示了当前宿主机上的防火墙的配置，增加了多个自定义的链。其中libvirt-host-in、libvirt-in、libvirt-in-post、libvirt-out是当libvirt第一次加载网络过滤器时自动创建的，而FI-vnet0、FO-vnet0、HI-vnet0是在初始化虚拟机centos6.10的网络接口时创建的(当前虚拟机centos6.10在宿主机网卡上的名称为vnet0)。

libvirt通过调用系统的iptables、ip6tables和ebtables命令完成防火墙的配置，这可以通过查看系统日志/var/log/messages得到印证。

可以将一个网络过滤器与多台虚拟机相关联，从而实现共享配置。例如：修改虚拟机win2k3的配置文件，并且使用网络过滤器blockicmp，这样也可以阻止它的ICMP流量。

下面清除实验环境，命令如下：

```
13 #virsh shutdown centos6.10

14 #virsh edit centos6.10
    删除<filterref filter='blockicmp'/>

15 #virsh nwfilter-undefine blockicmp
   Network filter blockicmp undefined
```

只能删除未使用的网络过滤器。第13行命令先关闭虚拟机，然后编辑配置文件，删除<filterref filter='blockicmp'/>。最后使用第15行命令取消网络过滤器的定义。

5.11.2 网络过滤器的管理工具

目前还不能通过Cockpit、virt-manager来管理网络过滤器，所以我们主要通过virsh中nwfilter-开头的子命令进行维护。

(1) nwfilter-define：根据XML文件定义或更新网络过滤器。

(2) nwfilter-dumpxml：输出 XML 格式的网络过滤器信息。
(3) nwfilter-edit：编辑网络过滤器的 XML 配置文件。
(4) nwfilter-list：列出网络过滤器清单。
(5) nwfilter-undefine：取消网络过滤器的定义。
(6) nwfilter-binding-create：根据 XML 文件创建网络过滤器的绑定。
(7) nwfilter-binding-delete：删除网络过滤器的绑定。
(8) nwfilter-binding-dumpxml：输出 XML 格式的网络过滤器所绑定的信息。
(9) nwfilter-binding-list：列出网络过滤器绑定的清单。
执行的命令如下：

```
1 # virsh help filter
  Network Filter (help keyword 'filter'):
   nwfilter-define           define or update a network filter from an XML file
   nwfilter-dumpxml          network filter information in XML
   nwfilter-edit             edit XML configuration for a network filter
   nwfilter-list             list network filters
   nwfilter-undefine         undefine a network filter
   nwfilter-binding-create create a network filter binding from an XML file
   nwfilter-binding-delete delete a network filter binding
   nwfilter-binding-dumpxml network filter information in XML
   nwfilter-binding-list    list network filter bindings

2 # virsh nwfilter-define --help
  NAME
    nwfilter-define - define or update a network filter from an XML file

  SYNOPSIS
    nwfilter-define <file>

  DESCRIPTION
    Define a new network filter or update an existing one.

  OPTIONS
    [--file] <string> file containing an XML network filter description
```

5.11.3　预安装的网络过滤器

在 RHEL/CentOS 8 中安装虚拟化组件时，会自动安装一些网络过滤器。与自定义的网络过滤器一样，都保存在/etc/libvirt/nwfilter/目录中。

这些过滤器的命名很直观，可以从名称猜测出其大概的用途。详细的描述如表 5-14 所示。

表 5-14 预安装的网络过滤器

名称	描 述
no-arp-spoofing	防止虚拟机欺骗 ARP 流量；该过滤器仅允许 ARP 请求和答复消息，并强制这些数据包包含虚拟机的 MAC 和 IP 地址
allow-arp	允许双向 ARP 流量
allow-ipv4	允许双向 IPv4 流量
allow-ipv6	允许双向 IPv6 流量
allow-incoming-ipv4	允许传入的 IPv4 流量
allow-incoming-ipv6	允许传入的 IPv6 流量
allow-dhcp	允许虚拟机通过任何 DHCP 服务器获得 IP 地址
allow-dhcpv6	与 allow-dhcp 类似，但用于 DHCPv6
allow-dhcp-server	允许虚拟机从指定的 DHCP 服务器获得 IP 地址。必须对此过滤器提供 DHCP 服务器的 IPv4 地址。变量的名称必须为 DHCPSERVER
allow-dhcpv6-server	与 allow-dhcp-server 类似，但用于 DHCPv6
no-ip-spoofing	防止虚拟机发送源 IP 地址与数据包中的源 IP 地址不同的 IPv4 数据包
no-ipv6-spoofing	类似于 no-ip-spoofing，但用于 IPv6
no-ip-multicast	防止虚拟机发送 IP 组播报文
no-ipv6-multicast	类似于 no-ip-multicast，但用于 IPv6
clean-traffic	防止 MAC、IP 和 ARP 欺骗。该过滤器引用其他几个过滤器作为构建块

上述大多数网络过滤器只是构建块，需要与其他网络过滤器结合使用以提供有用的网络流量过滤。下面我们分析一下网络过滤器 clean-traffic，它引用了多个其他的网络过滤器，代码如下：

```
1 # virsh nwfilter - dumpxml clean - traffic
  < filter name = 'clean - traffic' chain = 'root'>
    < uuid > 1a4aadd7 - 62c0 - 47fe - a91b - 6becc51fa549 </uuid >
    < filterref filter = 'no - mac - spoofing'/>
    < filterref filter = 'no - ip - spoofing'/>
    < rule action = 'accept' direction = 'out' priority = ' - 650'>
      < mac protocolid = 'ipv4'/>
    </rule >
    < filterref filter = 'allow - incoming - ipv4'/>
    < filterref filter = 'no - arp - spoofing'/>
    < rule action = 'accept' direction = 'inout' priority = ' - 500'>
      < mac protocolid = 'arp'/>
    </rule >
    < filterref filter = 'no - other - l2 - traffic'/>
    < filterref filter = 'qemu - announce - self'/>
  </filter >
```

第 1 行命令的输出是这个名为 clean-traffic 过滤器的 XML 定义。它引用了 no-mac-

spoofing、no-ip-spoofing、allow-incoming-ipv4、no-arp-spoofing、no-other-l2-traffic、qemu-announce-self 等 6 个子过滤器，还有两个流量过滤规则。clean-traffic 过滤器的结构如图 5-46 所示。

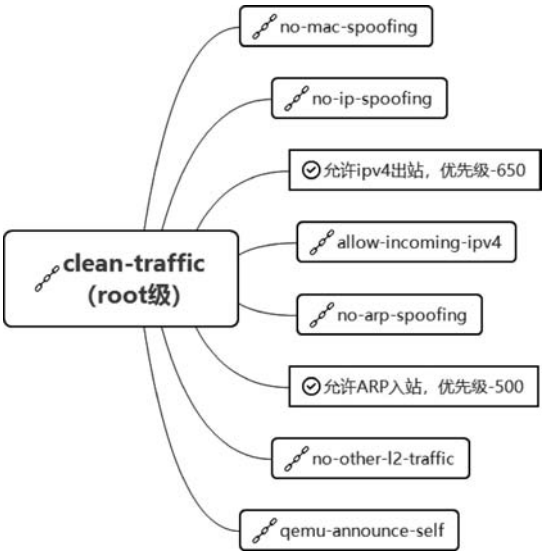

图 5-46　clean-traffic 过滤器结构

接下来执行的命令如下：

```
2 # virsh nwfilter - dumpxml no - mac - spoofing
  < filter name = 'no - mac - spoofing' chain = 'mac' priority = ' - 800'>
    < uuid > 5036c658 - 6bea - 4299 - aff4 - 0bd53a9048dd </uuid >
    < rule action = 'return' direction = 'out' priority = '500'>
      < mac srcmacaddr = ' $ MAC'/>
    </rule >
    < rule action = 'drop' direction = 'out' priority = '500'>
      < mac/>
    </rule >
  </filter >

3 # virsh nwfilter - dumpxml no - ip - spoofing
  < filter name = 'no - ip - spoofing' chain = 'ipv4 - ip' priority = ' - 710'>
    < uuid > 724ec182 - c6fe - 4289 - a1d3 - 8b0b4821a897 </uuid >
    < rule action = 'return' direction = 'out' priority = '100'>
      < ip srcipaddr = '0.0.0.0' protocol = 'udp'/>
    </rule >
    < rule action = 'return' direction = 'out' priority = '500'>
      < ip srcipaddr = ' $ IP'/>
    </rule >
```

```
    <rule action = 'drop' direction = 'out' priority = '1000'/>
  </filter>
```

第 2 行命令输出了网络过滤器 no-mac-spoofing 的定义。第 3 行命令输出了网络过滤器 no-ip-spoofing 的定义。它们仅包含了流量过滤规则，而没有包含其他网络过滤器的引用。

提示：虚拟机的使用者有可能通过修改 IP 和 MAC 地址的方式来伪造主机身份，这给网络安全防护带来了巨大的挑战。可以通过配置网络过滤器 clean-traffic 来检查出站的流量中的 IP 及 MAC 地址。如果不是宿主机分配给虚拟机的 IP 和 MAC 地址，则认为是伪造的地址从而可以进行阻断。

5.11.4 网络过滤器语法基本格式

与 libvirt 管理的其他对象一样，网络过滤器也使用 XML 格式配置。从高层次看，格式如下：

```
<filter name = '过滤器名称' chain = '链的类型'>
  <uuid>链的 UUID</uuid>

  <rule ...>
    ....
  </rule>

  <filterref filter = 'XXXX'/>
</filter>
```

每个网络过滤器都有一个名称和 UUID，它们用作唯一标识符。网络过滤器可以具有 0 个或多个<rule>元素，它们用于定义网络控制。网络过滤器还可以通过 filterref 来引用其他的网络过滤器。

1. 链的类型

可以将过滤规则组织成链，从而形成树形结构。数据包从 root 链中开始进入评估流程，然后沿着分支链继续对其评估。既可以从这些分支链返回上一级链中，也可以被过滤规则直接丢弃或接收从而结束评估。

当激活网络过滤器时，libvirt 会自动为每个虚拟机的网络接口创建独立的网络过滤的 root 链。用户创建自定义的网络过滤器时，默认的类型是 root，也可以指定链的类型。

chain 是一个可选属性。使用它可以更好地组织过滤器，从而可以被防火墙子系统进行更有效的处理。它必须是以 mac、vlan、stp、arp、rarp、ipv4 或 ipv6 开头的。

示例 XML 文件如下：

```
#grep -h "chain" /etc/libvirt/nwfilter/*.xml
  <filter name='allow-arp' chain='arp' priority='-500'>
  <filter name='allow-dhcp-server' chain='ipv4' priority='-700'>
  <filter name='allow-dhcp' chain='ipv4' priority='-700'>
  <filter name='allow-incoming-ipv4' chain='ipv4' priority='-700'>
  <filter name='allow-ipv4' chain='ipv4' priority='-700'>
  <filter name='clean-traffic-gateway' chain='root'>
  <filter name='clean-traffic' chain='root'>
  <filter name='no-arp-ip-spoofing' chain='arp-ip' priority='-510'>
  <filter name='no-arp-mac-spoofing' chain='arp-mac' priority='-520'>
  <filter name='no-arp-spoofing' chain='root'>
  <filter name='no-ip-multicast' chain='ipv4' priority='-700'>
  <filter name='no-ip-spoofing' chain='ipv4-ip' priority='-710'>
  <filter name='no-mac-broadcast' chain='ipv4' priority='-700'>
  <filter name='no-mac-spoofing' chain='mac' priority='-800'>
  <filter name='no-other-l2-traffic' chain='root'>
  <filter name='no-other-rarp-traffic' chain='rarp' priority='-400'>
  <filter name='qemu-announce-self-rarp' chain='rarp' priority='-400'>
  <filter name='qemu-announce-self' chain='root'>
```

2. 过滤规则

下面是一个简单的网络过滤器示例。它有这样一个规则，如果入站 IP 数据包中源 IP 地址是 192.168.122.21，则丢弃此数据包，示例代码如下：

```
<filter name='blockip' chain='ipv4'>
  <uuid>ebcdaeg1-a35f-62be-124e-208dadf2345a</uuid>
  <rule action='drop' direction='in' priority='500'>
    <ip match='yes' srcipaddr='192.168.122.21'/>
  </rule>
</filter>
```

rule 元素可包含如下属性。

(1) action：强制性的属性。必须是 drop、reject、accept、return 或 continue 中的一种。其中 drop 匹配规则无须进一步分析就静默丢弃数据包；reject 匹配规则将生成没有进一步分析结果的 ICMP 拒绝消息；accept 匹配规则接收无须进一步分析的数据包；return 匹配规则通过此过滤器，但将控制权返回给调用过滤器以进一步分析；continue 匹配规则继续到下一个规则以进一步分析。

(2) direction：强制性的属性。必须为 in、out 或者 inout 中的一种，分别针对入站、出站或出入站的流量。

(3) priority：可选的属性。它控制着规则相对于其他规则的实例化顺序。值较低的规则将在值较高的规则之前实例化。有效值的范围为－1000～1000，默认值是 500。

(4) statematch：可选的属性。如果值是 0 或 false，则会关闭基于连接状态匹配。默认值为 true。

在本示例中的<rule action='drop' direction='in' priority='500'>，表示对入站的流量进行 drop 操作，其优先级为 500。

在 rule 中，需要设置与特定协议相关的匹配条件。一般模式是：

```
<协议名称 match='yes|no' attribute1='value1' attribute2='value2'/>
```

目前支持的协议包括 mac、arp、rarp、ip、ipv6、tcp/ip、icmp/ip、igmp/ip、udp/ip、udplite/ip、esp/ip、ah/ip、sctp/ip、tcp/ipv6、icmp/ipv6、igmp/ipv6、udp/ipv6、udplite/ipv6、esp/ipv6、ah/ipv6、sctp/ipv6。

match 的默认值是 yes。

在本示例中的<ip match='yes' srcipaddr='192.168.122.21'/>，表示“如果源地址匹配 192.168.122.21”的过滤条件。

libvirt 所支持的协议及属性的表示方法，可参见 https://libvirt.org/formatnwfilter.html。

3. 变量的使用

为了提高网络过滤器的灵活性，可以在其中使用变量来替换固定的值。例如可以将上述示例中的网络过滤器进行修改，示例代码如下：

```
<filter name='blockip' chain='ipv4'>
  <uuid>ebcdaeg1-a35f-62be-124e-208dadf2345a</uuid>
  <rule action='drop' direction='out' priority='500'>
    <ip match='no' srcipaddr='$IP'/>
  </rule>
</filter>
```

示例中的<ip match='no' srcipaddr='$IP'/>使用了一个名称为 IP 的变量。引用的变量始终要以 $ 符号为前缀。变量值的格式需要与属性的类型相匹配，例如 IP 参数必须包含点分十进制格式的 IP 地址。如果格式不正确，将会造成网络过滤器无法实例化，从而导致无法启动虚拟机或网络接口。

虚拟机 XML 文件引用这些过滤器的时候，可以提供变量名和值。下面的示例中，虚拟机在调用过滤器 blockip 时，提供了 3 次变量名 IP 和 IP 地址。libvirt 会创建 3 个单独的过滤规则，每个 IP 地址对应一个。这样会阻断来自这 3 个 IP 地址主机的访问。

示例代码如下：

```
...
<devices>
  <interface type='bridge'>
    <mac address='0a:1b:2c:3d:4e:5f'/>
```

```
      <filterref filter = 'blockip'>
        <parameter name = 'IP' value = '192.168.122.21'/>
        <parameter name = 'IP' value = '192.168.122.22'/>
        <parameter name = 'IP' value = '192.168.122.23'/>
      </filterref >
    </interface >
  </devices >
  …
```

网络过滤器有一些保留的变量，其名称及含义如表 5-15 所示。

表 5-15　网络过滤器的保留变量

变量名称	含　义
MAC	接口的 MAC 地址
IP	接口使用的 IP 地址列表
IPV6	接口使用的 IPV6 地址列表
DHCPSERVER	受信任的 DHCP 服务器的 IP 地址列表
DHCPSERVERV6	受信任的 DHCP 服务器的 IPv6 地址列表
CTRL_IP_LEARNING	IP 地址检测方式的选择

最常用的保留变量是 MAC 和 IP，在预安装的网络过滤器中使用了它们。

变量 MAC 表示网络接口的 MAC 地址。如果未显式地提供，则 libvirt 守护程序会自动将其设置为网络接口的 MAC 地址。

变量 IP 表示虚拟机内部的操作系统应在给定接口上使用的 IP 地址。如果未显式地提供，则 libvirt 守护程序将尝试确定接口上正在使用的 IP 地址。

提示：如果网络过滤器中使用变量 IP 但未为其分配任何值，则将自动激活 libvirt 对虚拟机接口上使用的 IP 地址的检测。可以通过变量 CTRL_IP_LEARNING 来控制 IP 地址的学习方法，其有效值为 any、dhcp 或 none。默认值是 any，它表示 libvirt 可以使用任何数据包来确定虚拟机正在使用的地址。

5.11.5　自定义网络过滤器示例

下面我们通过示例来学习自定义过滤器的创建。假设现在要构建一个满足以下要求的网络过滤器：

(1) 防止虚拟机的 MAC、IP 和 ARP 欺骗。

(2) 仅打开虚拟机网络接口的 TCP 22 和 80 端口。

(3) 允许虚拟机向外发送 ping 流量，但不允许从外部 ping 虚拟机。

(4) 允许虚拟机执行 DNS 查询(UDP 53 的出站)。

示例 XML 文件如下：

```
<filter name='testfilter1'>
  <!-- 引用 clean-traffic 以防止 MAC、IP 和 ARP 欺骗. 不提供 MAC 和 IP 地址，由 libvirt 自动探查
出 VM 的实际值 -->
  <filterref filter='clean-traffic'/>

  <!-- 允许 TCP 端口 22(ssh)的入站 -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <!-- 允许 TCP 端口 80(http)的入站 -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!-- 允许出站的 ICMP echo 请求 -->
  <rule action='accept' direction='out'>
    <icmp type='8'/>
  </rule>

  <!-- 允许入站的 ICMP echo 响应请求 -->
  <rule action='accept' direction='in'>
    <icmp type='0'/>
  </rule>

  <!-- 允许出站的 UDP DNS 查找 -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

  <!-- 丢弃其他所有流量 -->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>
</filter>
```

预安装的 clean-traffic 网络过滤器已满足防止欺骗的要求，因此可以直接引用它，然后通过前两条规则分别允许 TCP 端口 22 和 80 的入站，接下来的 2 条规则用于控制 ping，随后是允许 DNS 查询。为了禁止所有其他流量出入站，最后添加一条丢弃所有其他流量的规则。

这个过滤器被命令为 testfilter1，在虚拟机的网络接口 XML 中可以这样来引用它，代码如下：

```
[…]
      <interface type='network'><mac address='52:54:00:27:5f:c9'/>
       <source network='default'/>
       <model type='virtio'/>
       <filterref filter='testfilter1'/>
       <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
     </interface>
[…]
```

5.12　本章小结

本章讲解了虚拟网络的管理，包括隔离、NAT、桥接、路由、开放等网络类型的原理与配置，以及 VLAN 和网络过滤器的原理与配置。

第 6 章将讲解管理虚拟存储。